UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

TRABAJO FIN DE MÁSTER

Filogenómica del complejo *Pseudomonas aeruginosa*

Máster Universitario en Bioinformática y Biología Computacional

Autor: Pérez Redondo, Sara

Tutor: Redondo Nieto, Miguel
Departamento de Biología /Facultad de ciencias

FECHA: Septiembre, 2018

Sumario

Resumen

El complejo de *P. aeruginosa* es un conjunto de bacterias pertenecientes al orden Pseudomonadales, unas γ-proteobacterias gram negativas que se encuentran en distintos nichos ecológicos y que está compuesto por una gran variedad de especies.

A lo largo de los años, las especies se han definido en base a su morfología y sus actividades metabólicas. Después mediante la secuenciación de relojes filogenéticos como el 16S y por hibridación DNA-DNA para comparaciones de especies o cepas dos a dos, pero estas técnicas se están sustituyendo por métodos de caracterización genotípica utilizando una serie de genes marcadores que se encuentran en todas las especies. En esto se basan los Análisis Multilocus de Secuencia (MSLA en inglés), aunque existen otros métodos basados en el estudio del genoma completo para obtener filogenias más precisas que las que se obtienen mediante MLSA, tales como la identidad media de nucleótidos (Average Nucleotide Identity, ANI), ANIb (ANI como una identidad media de todas las coincidencias de blastn) y ANIm (ANI con el software MUMmer).

En este trabajo se estudiaron las relaciones filogenéticas de las especies pertenecientes al complejo de *P.aeruginosa* utilizando MLSA y ANIb. Al mismo tiempo, también se definió el genoma central del complejo, el genoma principal de cada grupo, el genoma específico de cada especie, el genoma específico de cada grupo y el pangenoma gracias al estudio de los ortólogos, así como las distintas características funcionales que definan a cada grupo de P. aeruginosa.

Introducción

Las bacterias pertenecientes al orden *Pseudomonadales*, son unas γ-proteobacterias gram negativas que se encuentran en una gran variedad de ambientes y que tienen una gran versatilidad en la utilización de compuestos orgánicos como energía y compuestos de carbono (1,2), y la producción de metabolitos secundarios (3,4). *Pseudomonas* es uno de los géneros más diversos y más complejos, ya que se han descrito hasta la fecha más de 250 especies (5). En algunos estudios, se han descrito una serie de grupos y subgrupos en los que se divide este género (6,7), entre los cuales se incluye el grupo de *Pseudomonas aeruginosa*, dividido también a su vez en subgrupos. Dentro de este grupo de *P. aeruginosa*, se han observado especies que confieren una resistencia a hongos (8) y microbios (9) a las plantas que infectan, liberando una serie de metabolitos secundarios y factores de virulencia, los cuales pueden influir también en el crecimiento de la planta (9,10). Al mismo tiempo, estas bacterias también tienen una gran relevancia en Salud Pública, ya que están involucradas en una serie de infecciones nosocomiales debido a su gran resistencia a antibióticos (11,12). Una de las enfermedades más comunes en las que está implicada esta bacteria es la fibrosis quística, una enfermedad respiratoria en la que *P. aeruginosa* crea una serie de biofilms para desarrollar su actividad en ellos; aunque también se han observado infecciones en unidades de quemados (13) y en multitud de pacientes con heridas (14).

El grupo *P.aeruginosa* está compuesto por una gran variedad de especies, entre las que se incluyen *P. oleovorans, P. nitroreducens y P. mendocina*, entre otras. Debido a la gran diversidad de este grupo, y a la gran variedad de especies de *P. aeruginosa* dentro del grupo, es necesario realizar una revisión de la clasificación taxonómica del mismo. Además, en este 2018, se han descrito nuevas especies, por ejemplo, *P. furukawaii*, que también está incluida dentro del grupo de P. *aeruginosa*. Al ser un grupo muy complejo, es necesario estudiar su filogenia para poder tener un mayor conocimiento sobre las especies que se incluyen en él y saber qué diferencias existen entre ellas, sobre todo por su relación con las infecciones en humanos. De esta manera se podrían desarrollar nuevos fármacos o nuevas formas de controlar los brotes nosocomiales de estas bacterias.

Los primeros estudios de taxonomía de este grupo de bacterias comenzaron con la definición del género *Pseudomonas* en 1894 por Migula, basándose en sus características morfológicas (15), pero posteriormente se empezaron a basar también en sus características bioquímicas, fisiológicas y nutricionales para distinguir unas bacterias de otras por diferencias fenotípicas no relacionadas (16). A continuación, se desarrollaron las técnicas de hibridación y se realizó una revisión de la taxonomía de este grupo gracias a la hibridación DNA-DNA (DDH) (17) y rRNA-DNA (18), la cual estableció que el género *Pseudomonas* se dividía en 5 grupos.

Actualmente, estas técnicas de hibridación se están sustituyendo por métodos más avanzados de caracterización genotípica que permiten conocer de una forma más fideligna la filogenia bacteriana. A lo largo de estos años, se han utilizado una serie de genes marcadores que se encuentran en todas las especies, utilizándose en este trabajo algunos de ellos tales como: *gyrB, rpoB, rpoD* y *16S rDNA*. En muchos estudios, se analizó únicamente el gen 16S (secuencia de la subunidad pequeña del ribosoma), ya que es uno de los marcadores universales para la clasificación

bacteriana (19), pero se sabe que las bacterias que están muy relacionadas entre sí no se pueden diferenciar correctamente basándose únicamente en este gen (7). Por todo esto, se analizaron también las secuencias parciales de los genes denominados *housekeeping* o constitutivos, que son aquellos que se expresan siempre en las bacterias, como resultado de la interacción entre la ARN polimerasa y el promotor sin necesidad de regulación adicional (20). Algunos de estos genes marcadores son *gyrB*, que codifica la subunidad beta de la girasa, responsable del superenrrollamiento negativo del ADN durante la replicación (6), *rpoB* codifica para la subunidad beta de la ARN polimerasa y se ha establecido cono un buen marcador para el análisis filogenético y la identificación de bacterias (21) y por último, el gen *rpoD* que codifica la subunidad sigma 70 de la ARN polimerasa (22). Para un mejor análisis filogenómico, se ha observado que es mejor estudiar las secuencias de los cuatro genes constitutivos para facilitar la identificación de nuevas cepas y para establecer una correcta filogenia (23,24), en vez de utilizar un único gen marcador.

En esto se basan los Análisis Multilocus de Secuencia (MSLA en inglés), en analizar secuencias parciales de los genes (normalmente suelen ser constitutivos) para la caracterización de bacterias y microorganismos (25). El inconveniente que tiene este método es que, como se ha dicho para el rDNA 16S, las bacterias no se pueden clasificar correctamente, ya que no se pueden diferenciar bien las que están muy próximas. Gracias al desarrollo de las nuevas técnicas de secuenciación masiva de genomas, se fueron desarrollando otros métodos para el estudio filogenético de las especies basados en la secuenciación de genoma completo, los cuales permiten desarrollar filogenias más precisas que las que se obtienen mediante MLSA (26) y estudiar genomas borrador o *draft* que se encuentren en las bases de datos; por lo tanto es posible que estas técnicas también sustituyan a la DDH para el estudio taxonómico de las especies (27). Uno de estos métodos es la identidad media de nucleótidos (Average Nucleotide Identity, ANI), que representa una media de la identidad de las secuencias codificantes de dos genomas (es una comparación de dos genomas), obteniéndose una medida fiable para conocer la distancia evolutiva de las especies (27). Dos años después, se modificó este método, ya que simulaban la fragmentación del ADN propia de los experimentos de DDH, cortando artificialmente la secuencia del genoma en fragmentos de 1020 pb (28). Actualmente, se ha desarrollado un índice derivado de ANI, que calcula el ANI como una identidad media de todas las coincidencias de blastn (29) entre dos secuencias de genoma en las que sólo se tienen en cuanta las coincidencias con mínimo un 30% de identidad de secuencia global. Al mismo tiempo, también se ideó otro índice a partir de ANI, pero en este caso fue con el software MUMmer (30), el cuál permitía obtener resultados más rápidos, aunque éste (ANIm) se suele utilizar menos que ANIb. A la hora de relacionar DDH con ANI, el umbral de ANI que equivale con el 70% de DDH es el 95-96% (28,31), respaldado con un coeficiente de correlación de la frecuencia de tetranucleótidos (31).

En contrapartida, mientras el ANI es un índice de identidad, la filogenia basada en la distancia BLAST del genoma (Genome BLAST Distance Phylogeny, GBDP) es un índice de relación de genomas basado en la distancia existente entre ellas (32). En este caso, dos genomas se alinean entre sí utilizando BLAST (29), BLAST+ (33) y BLAT (34), y se obtiene la distancia entre los genomas. El servidor web que está basado en este índice, se llama calculadora de la distancia genoma a genoma (Genome-to-Genome Distance Calculator, GGDC) y recientemente se ha

mejorado para clarificar la taxonomía de las subespecies de procariotas (35). En este proyecto, en un principio se pensó utilizar GBDP para medir las distancias entre los genomas, pero al final no se utilizó puesto que todavía no han liberado el código para poder utilizarlo de forma local en el ordenador.

Como se puede observar, todos estos métodos (ANI, ANIb, ANIm y GBDP), utilizan el genoma completo de las bacterias, y así se comparan dos genomas fácilmente y de una manera objetiva y reproducible. Estos métodos se incluyen dentro del término *índices generales de relación genómica* (Overall Genome Relatedness Indices, OGRI).

En este proyecto, se estudiaron las relaciones filogenéticas de las especies pertenecientes al complejo de *P.aeruginosa* utilizando MLSA y ANIb. Al mismo tiempo, también se definió el genoma central del complejo, el genoma principal de cada grupo, el genoma específico de cada especie, el genoma específico de cada grupo y el pangenoma.

Objetivos

El objetivo principal de este TFM es conocer la taxonomía del grupo *P. aeruginosa* gracias a los análisis filogenómicos desarrollados en este trabajo, y compararlos con la taxonomía establecida en las bases de datos.

De este objetivo principal surgieron una serie de objetivos secundarios o específicos necesarios para conseguir el objetivo principal:

- Definir la estructura del grupo *P. aeruginosa* gracias a MLSA y ANIb.
- Analizar los datos filogenómicos del complejo mediante ANIb.
- Encontrar cuál es el genoma central, así como el que es específico de cada grupo.
- Analizar la distribución filogenética de distintos aspectos o funciones bacterianas.

Material y métodos

Datos de los genomas de *P. aeruginosa*

Los genomas pertenecientes al género *Pseudomonas* se descargaron del servidor FTP del NCBI (ftp.ncbi.nih.gov) en Junio de 2018, mediante el desarrollo del script ftp.py, que se recoge en el Anexo-I.

Una vez descargados los genomas, se obtuvieron una serie de carpetas en las que se incluían los siguientes archivos con sus características (https://www.ncbi.nlm.nih.gov/genome/doc/ftpfaq/):

- GCF_001086625.1_ASM108662v1_cds_from_genomic.fna.gz: archivo en formato FASTA de las secuencias nucleotídicas que corresponden a todas las CDS (secuencias codificantes) anotadas en el ensamblaje, basándose en la secuencia genómica.
- GCF_001086625.1_ASM108662v1_feature_count.txt.gz: Archivo de texto tabular que contiene el recuento de genes, ARN, CDS y características similares, en base a los datos incluidos en el archivo * _feature_table.txt.gz

- GCF_001086625.1_ASM108662v1_feature_table.txt.gz: Archivo de texto tabular que contiene ubicaciones y atributos para un subconjunto de características anotadas. Los tipos de características incluidas son: gen, CDS, ARN (todos los tipos), operón, región C / V / N / S y segmento V / D / J.
- GCF_001086625.1_ASM108662v1_genomic.fna.gz: Archivo FASTA de la secuencia genómica en el ensamblaje. El archivo genomic.fna.gz incluye cromosomas, plásmidos, orgánulos, scaffolds no localizados, scaffolds sin ubicar y cualquier loci alternativo.
- GCF_001086625.1_ASM108662v1_genomic.gbff.gz: Archivo GenBank de la secuencia genómica en el ensamblaje.
- GCF_001086625.1_ASM108662v1_genomic.gff.gz: Archivo GFF3 de la anotación de la secuencia genómica.
- GCF_001086625.1_ASM108662v1_protein.faa.gz: Archivo FASTA de los productos proteicos anotados en el ensamblaje del genoma.
- GCF_001086625.1_ASM108662v1_protein.gpff.gz: Archivo GenPept de los productos proteicos anotados en el ensamblaje del genoma.
- GCF_001086625.1_ASM108662v1_rna_from_genomic.fna.gz: Archivo FASTA de las secuencias de nucleótidos correspondientes a todas las características de ARN anotadas en el ensamblaje, basadas en la secuencia del genoma.
- GCF_001086625.1_ASM108662v1_translated_cds.faa.gz: Archivo FASTA que contiene las secuencias traducidas de la secuencia de nucleótidos proporcionada en el archivo * _cds_from_genomic.fna.gz.
- GCF_001086625.1_ASM108662v1_wgsmaster.gbff.gz: Archivo GenBank del maestro WGS para el ensamblaje (sólo presente si existe un registro maestro WGS para las secuencias en el ensamblaje).

Como se puede observar, estos archivos tenían nombres muy largos, por lo que se desarrolló un script llamado cambiar_nombre_id.py en el que se cambió los nombres de los archivos, incluyendo únicamente el identificador de esa bacteria (por ejemplo, GCF_001086625.1).

Filogenia basada en MLSA

Las secuencias parciales de los genes 16S rDNA, gyrB, rpoD y rpoB de los 107 cepas tipo de Pseudomonas, descritas en Mulet et al., 2010, se descargaron de la base de datos PseudoMLSA (http://www.uib.es/microbiologiaBD/Welcome.php).

La filogenia basada en MLSA del complejo de *P. aeruginosa* se compone de los siguientes pasos: Primero se obtienen las secuencias de los genes *gyrB, rpoB, rpoD y 16S rDNA* a partir de la anotación genómica, si estaban disponibles mediante el script de seq.py (Anexo-I), y si no lo

estaban, utilizar BLASTN sobre la secuencia genómica prueba_seqIO.py (Anexo-I), después se alinearon las secuencias de cada uno de los genes entre sí con el programa Clustal Omega (36).

Después de concatenar las secuencias de cada gen, se alinearon las secuencias con el programa Clustal Omega, escribiendo el siguiente comando, uno por cada gen:

```
clustalo -i [archivo_fasta_concatenado] -o [archivo_salida]
```

A modo de ejemplo, para alinear las secuencias del gen *gyrB*, se ejecutó lo siguiente:

```
clustalo -i ~/Escritorio/Sara_Perez/MLSA/fasta_mlsa_gyrB.txt -o alin_gyrB
```

El resultado de este comando fue un archivo en el que se incluían todos las secuencias de cada gen de cada genoma bacteriano alineados.

Una vez que se generaron los 4 archivos con las secuencias alineadas de los 4 genes, se desarrolló un script concat_MLSA.py (Anexo-I) para poder concatenar en un único archivo la secuencia del gen *gyrB, rpoB, rpoD* y *16S* de cada genoma, es decir, para juntar en un archivo las secuencias de los cuatro genes del genoma de *P. aeruginosa PA01*, por ejemplo.

Obteniéndose, al final, un archivo llamado mlsa_concat_all.fasta que contiene las secuencias concatenadas de los 4 genes, correspondientes a cada genoma. A la hora de construir el árbol filogenético en MEGA, los nombres existentes de cada bacteria en el archivo mlsa_concat_all.fasta son muy largos, con lo cual se creó una base de datos con el identificador de la especie y el nombre de la bacteria, para poder modificar estos identificadores por el nombre corto de la bacteria. La estructura de esta base de datos fue la siguiente:

GCF_000005845.2	E. coli K-12
GCF_000006765.1	P. aeruginosa PAO1
GCF_000014625.1	P. aeruginosa UCBPP-PA14
GCF_000016565.1	P. mendocina ymp
GCF_000017205.1	P. aeruginosa PA7
GCF_000026645.1	P. aeruginosa LESB58
GCF_000148745.1	P. aeruginosa 39016
GCF_000168335.1	P. aeruginosa PACS2
GCF_000172395.1	P. aeruginosa PAb1
GCF_000204295.1	P. mendocina NK-01
GCF_000226155.1	P. aeruginosa M18
GCF_000233495.1	P. aeruginosa sp 2
GCF_000258285.1	P. aeruginosa LCT-PA102

Figura 1. Extracto de la base de datos con los identificadores de NCBI y los nombres cortos de las especies.

A modo de ejemplo, en la imagen inferior se muestra la estructura de los encabezamientos de cada genoma con los genes concatenados.

```
>GCF_000005845.2_ASM584v2_NC_000913.3 Escherichia coli str. K-12 substr. MG1655, complete genome
------ATGTCGAATTCTTATGACTCCTCCAGTATCAAAGTCCTGAAAGGGCTGGATGCG
GTGCGTAAGCGCCCGGGTATGTATATCGGCGACACGGATGACGGCACCGGTCTGCACCAC
ATGGTATTCGAGGTGGTAGATAACGCTATCGACGAAGCGCTCGCGGGTCACTGTAAAGAA
ATTATCGTCACCATTCACGCCGATAACTCTGTCTCTGTACAGGATGACGGGCGCGGCATT
CCGACCGGTATTCACCCGGAAGAGGGCGTATCGGCGGCGGAAGTGATCATGACCGTTCTG
CACGCAGGCGGTAAATTTGACGATAACTCCTATAAAGTGTCCGGCGGTCTGCACGGCGTT
GGTGTTTCGGTAGTAAACGCCCTGTCGCAAAAACTGGAGCTGGTTATCCAGCGCGAGGGT
```

Figura 2. Extracto del archivo .fasta con los cuatro genes concatenados.

Como se puede observar, la primera línea corresponde con la descripción del genoma de la bacteria, en este caso es GCF_000005845.2_ASM584v2 NC_000913.3 Escherichia coli str. K-12 substr. MG1655, complete genome. Este nombre es demasiado largo, con lo cual se cambió la descripción por el nombre corto de la bacteria, por E. coli K-12. El script que se utilizó para estas modificaciones estaba incluido en el script concat_MLSA.py (Anexo-I).

Finalmente se construyó el árbol filogenético con el software MEGA (v.7) (37).

En este paso, se utilizó el archivo mlsa_concat_all_mod.fasta para introducirlo como datos en el programa MEGA, y se realizó un análisis de *Maximum Likelihood* (ML) utilizando el modelo Tamura-Nei, con 1000 repeticiones de bootstrap y con *Escherichia coli* de outgroup.

Esto generó un árbol que se visualizó y exportó utilizando este mismo software, para analizar los grupos en los que se divide este complejo.

Filogenia basada en el genoma completo

Para determinar las relaciones filogenéticas entre los distintos genomas del complejo de *P. aeruginosa*, se realizaron los cálculos de ANIb (cálculo medio de la identidad de nucleótidos implementado con BLAST), gracias a un script escrito en Python que se puede encontrar en GitHub (https://github.com/ctSkennerton/scriptShed/blob/master/calculate_ani.py). Para ello se ejecutó el siguiente comando:

```
python3 calculate_ani.py -o [carpeta_destino] [archivos_entrada] -m ANIb
```

Se obtuvo una matriz de similitud de genomas (imagen 1), en la cual se eliminaron los valores 'NA' y los encabezamientos, y se convirtió en una matriz de distancias mediante la fórmula 100-similitud de ANIb%, utilizándose el script matriz_anib.py (Anexo-I).

	GCF_000005845.2_genomic	GCF_000006765.1_genomic
GCF_000005845.2_genomic	NA	0.8346606383869426
GCF_000006765.1_genomic	0.8346606383869426	NA
GCF_000014625.1_genomic	0.8339429623539721	0.9640807067237203
GCF_000016565.1_genomic	0.8368469626895079	0.8296796235776048
GCF_000017205.1_genomic	0.8332255979314803	0.9217184252345024
GCF_000026645.1_genomic	0.8334858681022881	0.9691552892705123
GCF_000148745.1_genomic	0.7948057240106426	0.966810022265542
GCF_000168335.1_genomic	0.8297410152951112	0.9681990923386763
GCF_000172395.1_genomic	0.7961189233624373	0.986455047990752
GCF_000204295.1_genomic	0.839538869786988	0.8268955120724799
GCF_000226155.1_genomic	0.8345344860485255	0.9696961401468785

Figura 3. Matriz de similitud de ANIb obtenida mediante el script calculate_ani.py.

El programa MEGA, tiene una utilidad que permite construir árboles filogéneticos a partir de matrices de distancias, pero este archivo tiene una extensión .meg y una morfología compleja que se detalla en la imgane inferior.

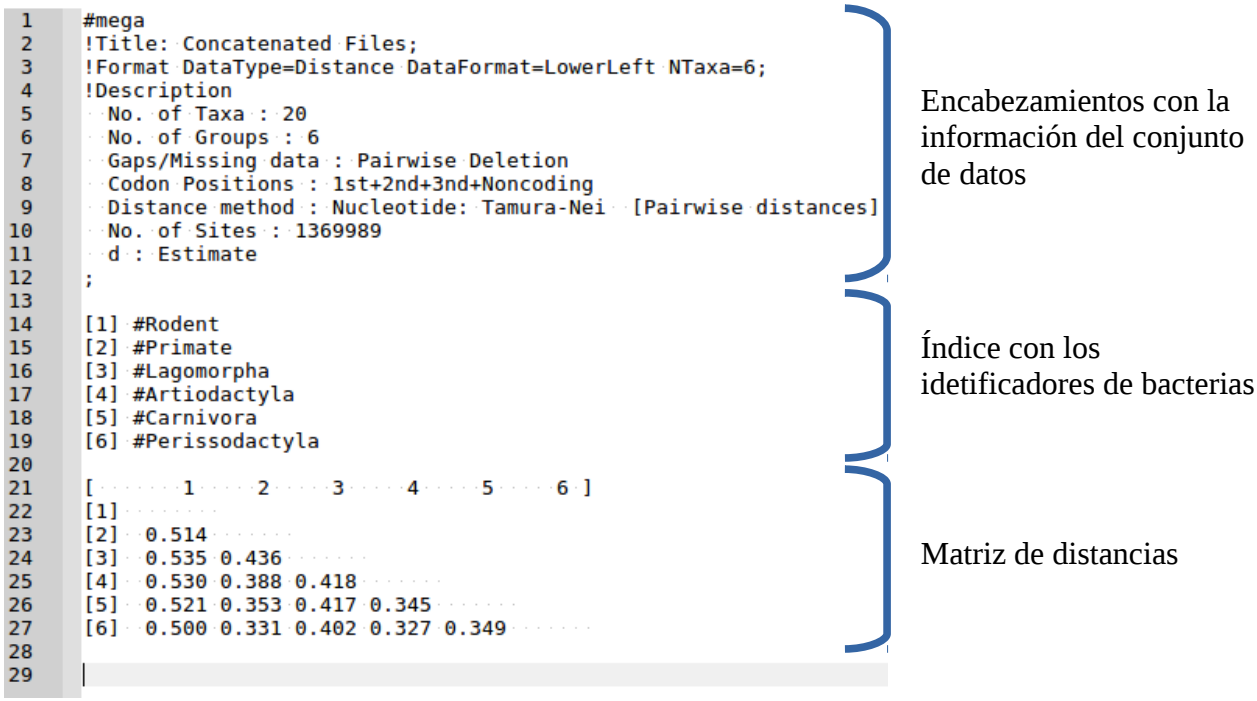

Figura 4. Estructura del archivo .meg necesario para construir el árbol ANIb.

Para poder convertir la matriz de distancias obtenida en el archivo .meg, se tuvo que desarrollar otro script crear_meg.py (Anexo-I) con dos partes diferenciadas, una en la que se crean los encabezamientos y el índice de los identificadores y otra en la que se crea la matriz de distancias con sus correspondientes cabeceras.

Una vez obtenido el archivo .meg, se cargó en el programa MEGA, se construyó el árbol Neighbor Joining y se visualizó, para poder estudiar los grupos existentes en este complejo.

Al mismo tiempo, con la matriz de similitud de ANIb obtenida anteriormente, se generó un *heatmap* para poder comprobar la clasificación del árbol de ANIb. Para ello, en primer lugar, se realizó un script heatmap.py que permite ordenar esta matriz en el mismo orden en el que aparecen las bacterias en el árbol ANIb.

Después se introdujo esa matriz en Rstudio (38), se generó el *heatmap* y se obtuvieron los distintos grupos en los que se clasificó el complejo, recuperándose también las especies que estaban en cada grupo.

Búsqueda y agrupación de ortólogos

Para ello, en primer lugar se identificaron los ortólogos de cada grupo dentro del complejo de *P. aeruginosa* utilizando la herramienta OrthoFinder (39), utilizando el siguiente comando:

```
orthofinder -f ./ -S diamond -t 8 -a 8
```

Donde -f es el directorio en el que se encuentran todas los archivos .faa con las proteínas, -S corresponde con el programa con el que se va a realizar la búsqueda de las secuencias (en este caso se utilizó diamond, puesto que es un programa que va mucho más rápido), -t que son el número de búsquedas que realiza a la vez y -a que son los análisis que realiza a la vez.

Cuando terminó este programa, se obtuvieron una serie de archivos temporales y archivos definitivos, de los cuales sólo se tuvieron en cuenta los siguientes archivos:

- Orthogroups.csv: fichero en formato tabular en el que están representados en cada fila los genes que pertenecen a un ortogrupo. Los genes de cada otrogrupo están organizados en columnas, una columna por especie. Por ejemplo, en la imagen inferior se muestra una imagen del formato de este archivo, en el que se observa que por cada ortogrupo (OG0000000), existen una serie de proteínas (NP_249480.1 amino acid permease...) separadas por comas pertenecientes a cada especie (GCF_000006765.1_ASM676v1_protein).

	A	B	
1		GCF_000006765.1_ASM676v1_protein	GCF_000014625.1_ASM1462v1_protein
2	OG0000000	NP_249480.1 amino acid permease [Pseudomonas aeruginosa PAO1], NP_249557.1	WP_003085884.1 MULTISPECIES_ aromatic
3	OG0000001	NP_248781.1 type VI secretion system protein VgrG [Pseudomonas aeruginosa PAO	WP_003139309.1 MULTISPECIES_ type VI s
4	OG0000002	NP_253186.1 ABC transporter [Pseudomonas aeruginosa PAO1], NP_253187.1 ABC t	WP_003096557.1 MULTISPECIES_ ABC trar
5	OG0000003	NP_249013.1 transporter [Pseudomonas aeruginosa PAO1], NP_250731.1 amino acid	WP_003088677.1 MULTISPECIES_ APC fam
6	OG0000004	NP_250128.1 two-component response regulator [Pseudomonas aeruginosa PAO1], N	WP_003083028.1 MULTISPECIES_ DNA-bin
7	OG0000005	NP_248913.1 hypothetical protein PA0222 [Pseudomonas aeruginosa PAO1], NP_249	WP_003095722.1 MULTISPECIES_ ABC trar
8	OG0000006	NP_248806.1 hypothetical protein PA0116 [Pseudomonas aeruginosa PAO1], NP_250	WP_003083753.1 MULTISPECIES_ transcrip
9	OG0000007	NP_251284.1 hypothetical protein PA2594 [Pseudomonas aeruginosa PAO1], NP_251	WP_003111308.1 MULTISPECIES_ sulfonate
10	OG0000008	NP_250245.1 cbb3-type cytochrome C oxidase subunit I [Pseudomonas aeruginosa P	WP_003087306.1 MULTISPECIES_ cytochro
11	OG0000009	NP_249575.1 C4-dicarboxylate-binding protein [Pseudomonas aeruginosa PAO1], NP_	WP_003094894.1 MULTISPECIES_ C4-dicarl
12	OG0000010	NP_250590.1 phenazine biosynthesis protein PhzA [Pseudomonas aeruginosa PAO1]	WP_003115119.1 MULTISPECIES_ phenazir
13	OG0000011	NP_250837.1 catalase HPII [Pseudomonas aeruginosa PAO1], NP_252926.1 catalase	WP_003094881.1 MULTISPECIES_ catalase
14	OG0000012	NP_252622.1 choline transporter [Pseudomonas aeruginosa PAO1], NP_253978.1 cho	WP_003096496.1 MULTISPECIES_ choline t
15	OG0000013	NP_249394.1 major facilitator superfamily transporter [Pseudomonas aeruginosa PAO	WP_003096941.1 MULTISPECIES_ MFS trar
16	OG0000014	NP_249396.1 alpha-1_6-rhamnosyltransferase MigA [Pseudomonas aeruginosa PAO1	WP_003140646.1 MULTISPECIES_ glycosyl
17	OG0000015	NP_249397.1 chloramphenicol acetyltransferase [Pseudomonas aeruginosa PAO1], N	WP_003111880.1 MULTISPECIES_ antibiotic

Figura 5. Estracto del archivo resultante del programa OrthoFinder, en el que se detallan los ortogrupos obtenidos.

- SpeciesTree_rooted.txt: fichero en formato newick en que se encuentra representado el árbol de las especies, con el que se observaron los grupos en los que se dividía este complejo.
- SingleCopyOrthogroups.txt: fichero en formato tabular en el que se encuentran los ortólogos con exactamente un gen en cada especie. Estos ortogrupos son muy útiles para inferir un árbol de especies.
- Statistics_Overall.csv: fichero csv en el que se recogen todas las estadísticas pertenecientes a los ortogrupos, tales como el número de ortogrupos que se encuentran en todas las especies (genoma central), los específicos de especie, etc.

Para estudiar los ortogrupos que existen en el complejo de *P. aeruginosa*, se cambió el nombre de las especies por un nombre más corto (por ejemplo, cambiando GCF_000006765.1_ASM676v1_protein por GCF_000006765.1), procesándose el archivo Orthogroups.csv con el script cambiar_nombre.py (Anexo-I).

Después de cambiar los nombres de los identificadores de especie, se creó otro script llamado csv2sql1.py (Anexo-I) en el que se recogieron en columnas el ortogrupo, las proteínas incluidas en ese ortogrupo y la especie a la que pertenecían, generándose un archivo en el que en la primera columna se encuentra el identificador del ortogrupo, en la segunda las proteínas que están incluidas en ese ortogrupo y en la tercera la especie a la que pertenecen.

Pero existe un problema, ya que para poder procesar esta tabla en cualquier programa de base de datos, en este caso LibreOffice Base, se tuvieron que separar las proteínas que existían en cada ortogrupo, generando una línea por cada proteína. Esto se realizó con el script csv2sql2.py (Anexo-I), generándose un archivo con el ortogrupo en la primera columna, el identificador de la proteína y el de la especie.

Una vez obtenido este archivo se quitaron los *missing values*, en este caso representados con 'nan', que se eliminaron, ya que no aportaban información para el estudio de los ortólogos, al mismo tiempo que también se acortaron los identificadores de la proteína, ya que a la hora de procesarlos con LibreOffice Base permite que las consultas sean más rápidas y más fáciles. Todo esto se realizó gracias al script csv2sql3.py (Anexo-I).

De esta manera se obtuvo el archivo que se utilizó como tabla en la base de datos de LibreOffice Base, para poder analizar el genoma central del grupo de *P. aeruginosa*, el específico de cada grupo.

Para introducir esta tabla en la base de datos de LibreOffice Base, se creó la tabla con *mysql*, *y* se utilizó un script llamado *csv2SQL* (https://github.com/najiji/csv2SQL), que te convierte cada fila en una orden de SQL para introducir los datos en la tabla creada.

A continuación, se creó otra tabla en la que se recogían las especies y los grupos a los que pertenecían tanto en el árbol MLSA como en el árbol de ANIb. También se introdujo esta tabla en *mysql*, de igual manera que la anterior, para poder analizar los ortogrupos pertenecientes al genoma central y los que son específicos de grupo, mediante la realización de una serie de consultas en SQL sobre esta tabla y la del párrafo anterior.

Una vez obtenidos esos tipos de ortogrupos, mediante scripts de Python se extrajeron las proteínas que se encontraban dentro de esos ortogrupos, para estudiar las secuencias que componen el genoma central y las específicas de grupo, gracias al script prot_comunes_grupo.py (Anexo-I) que se utilizó para extraer las proteínas ortólogas dentro de cada grupo.

Finalmente, se obtuvieron las secuencias ortólogas específicas de cada grupo mediante el script prot_diferenciales.py (Anexo-I) y se obtuvo la curva coleccionista para estudiar las fluctuaciones de secuencias ortólogas en el genoma central, mediante el script de Python muestreo_aleatorio.py para realizar un muestreo aleatorio de hasta 200 repeticiones de cepas y para representarlo después con R, gracias al script collector_curve.R. Esto se analizó, por ejemplo, para poder saber si, a medida que se descubren nuevos genomas de bacterias, el pangenoma aumenta o ya se han descubierto todas las bacterias necesarias para explicar el pangenoma del complejo de *P. aeruginosa.*

Estudio de las características funcionales

Para estudiar las características funcionales, se dividieron las especies en los grupos que se habían definido de los resultados del árbol ANIb, y se escogieron tres especies de cada grupo (salvo en el grupo de *P. jinjuensis* y *P.flavescens* para analizar qué genes estaban enriquecidos y de esta

manera poder saber las características funcionales que describen a cada grupo, gracias a la librería topGO escrita en R.

Para ello, en primer lugar se creó una tabla con la siguiente estructura:

Nombre_especie	Id_especie	Grupo_ANIb	Arch_output_R
P.aeruginosa_608748	GCF_001910045.1	1	esp_1_1_topGO.txt
P.aeruginosa_C2773C	GCF_000705175.1	1	esp_1_2_topGO.txt
P.aeruginosa_ATCC_9027	GCF_002601815.1	1	esp_1_3_topGO.txt
P.jinjuensis_NBRC_103047	GCF_002091655.1	2	esp_2_1_topGO.txt
P.jinjuensis_JCM_21621	GCF_900103845.1	2	esp_2_2_topGO.txt
P.citronellolis_SJTE_3	GCF_001654435.1	3	esp_3_1_topGO.txt
P.nitroreducens_NBRC_12694	GCF_002091755.1	3	esp_3_2_topGO.txt
P.nitroreducens_HBP1	GCF_000518065.1	3	esp_3_3_topGO.txt
P.alcaligenes_MRY13_0052	GCF_000474255.1	4	esp_4_1_topGO.txt
P.resinovorans_DSM_21078	GCF_000423545.1	4	esp_4_2_topGO.txt
P.alcaligenes_NBRC_14159	GCF_000467105.1	5	esp_5_1_topGO.txt
P.alcaligenes_NEB_585	GCF_001597285.1	5	esp_5_2_topGO.txt
P.alcaligenes_RU36E	GCF_900156135.1	5	esp_5_3_topGO.txt
P.flavescens_NBRC_103044	GCF_002091575.1	6	esp_6_1_topGO.txt
P.flavescens_LMG_18387	GCF_900100535.1	6	esp_6_2_topGO.txt
P.mendocina_DLHK	GCF_000287395.1	7	esp_7_1_topGO.txt
P.pseudoalcaligenes_AD6	GCF_000626735.1	7	esp_7_2_topGO.txt
P.oleovorans_RS1	GCF_900109155.1	7	esp_7_3_topGO.txt

Figura 6. Extracto del archivo en el que se incluyen el nombre de la bacteria, su identificador del NCBI, el grupo de ANIb al que pertenecen y el archivo donde se vna a guardar las características funcionales de cada especie.

En la cual se puede observar que la primera columna pertenece al nombre de la especie, la segunda a su identificador, la tercera es el grupo al que pertenecen en el árbol ANIb y la cuarta el archivo en que se guardaron las tablas que se generaron en R. Cabe destacar que en el nombre de los archivos de output de R, también se hizo alusión a si esos archivos incluían los resultados de procesos biológicos (Biological Process, BP), función molecular (Mollecular Function, MF) o componente celular (Cellular Compound, CC), creándose por ejemplo en la primera especie: esp_1_1_topGO_BP.txt, esp_1_1_topGO_MF.txt y esp_1_1_topGO_CC.txt.

A continuación, se ejecutó el script en R de topGO.R para estudiar las características funcionales de las distintas especies, preparando las listas de genes anotados y la lista de los genes interesantes, que se corresponden con los genes cuyas secuencias son ortólogas en todas las especies del grupo.

Estas listas de genes interesantes se extrajeron gracias a un script desarrollado en Python llamado genes_interesantes.py (Anexo-I), que permitía obtener estos genes a partir de las tablas conseguidas mediante las consultas de SQL.

Resultados y discusión

Filogenia basada en MLSA

En el análisis de MLSA obtenido gracias a la concatenación de las secuencias parciales de los genes *gyrB*, *rpoB*, *rpoD* y *16S rDNA*, generó un árbol en el cual se establecieron los grupos

existentes dentro del complejo de *P. aeruginosa*, utilizando como *outgroup* a *E. coli* y a *P. fluorescens*.

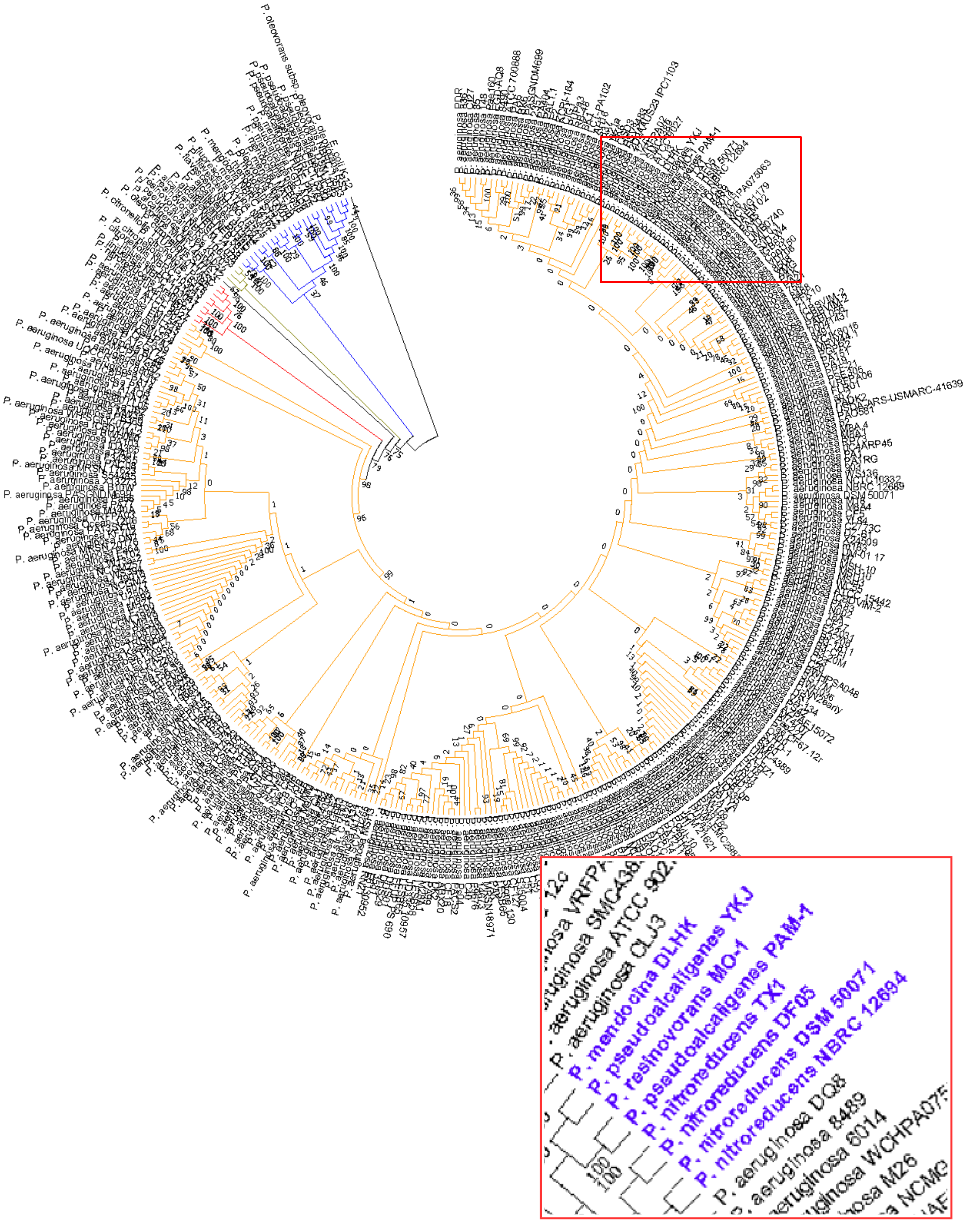

Figura 7. Árbol basado em MLSA.

Como se puede observar en la Fig.7, se obtuvieron cuatro grupos diferenciados, uno de los cuales es el propio grupo de *P. aeruginosa*, que es el más distal y más diverso (en naranja), y otros tres grupos que se analizarán en la Figura 8, puesto que al ser tan grande este grupo, no se pueden distinguir bien a simple vista. En este grupo, aparte de estar todas las especies de *P. aeruginosa*, también se encuentra un pequeño grupo en el que se incluyen otras especies como son *P. mendocina*, *P. pseudoalcaligenes*, *P. resinovorans y P. nitroreducens.* Esto podría deberse a que los genes que se utilizaron para realizar esta clasificación, son muy similares a los de *aeruginosa* y que por eso aparecen entre ellas.

Al haber tantos genomas, se comprimió el grupo de *P. aeruginosa* para poder clarificar el árbol y se obtuvo el que se recoge en la Figura 8, estableciéndose otros tres grupos, *P. citronellolis*, *P. resinovorans y P. mendocina*, aunque este último a su vez se podría clasificar en 4 subgrupos, *P. alcaligens, P. flavescens, P. mendocina, y P. pseudoalcaligenes.* En el recuadro rojo, se muestra una cepa de *P. oleovorans MOIL14HWK12* que debería estar junto con las otras cepas de *oleovorans* que se encuentran al final del árbol (subgrupo *P. pseudoalcaligenes*), pero no lo está. Esto podría deberse a que alguno de los genes que se utilizaron para este análisis esté mal anotado o que esté mutado y no se parezca a los existentes en los demás *oleovorans*.

Al mismo tiempo, en el rectángulo azul está señalado *P. fluorescens*, que se utilizó como *outgroup* para analizar la taxonomía de este complejo, pero en este caso está junto con las bacterias de *P. flavescens*, por lo cual, podría ser que alguno de estos genes *gyrB*, *rpoB*, *rpoD* o *16S rDNA*, sea muy similar a *flavescens* y que por esto sean similares.

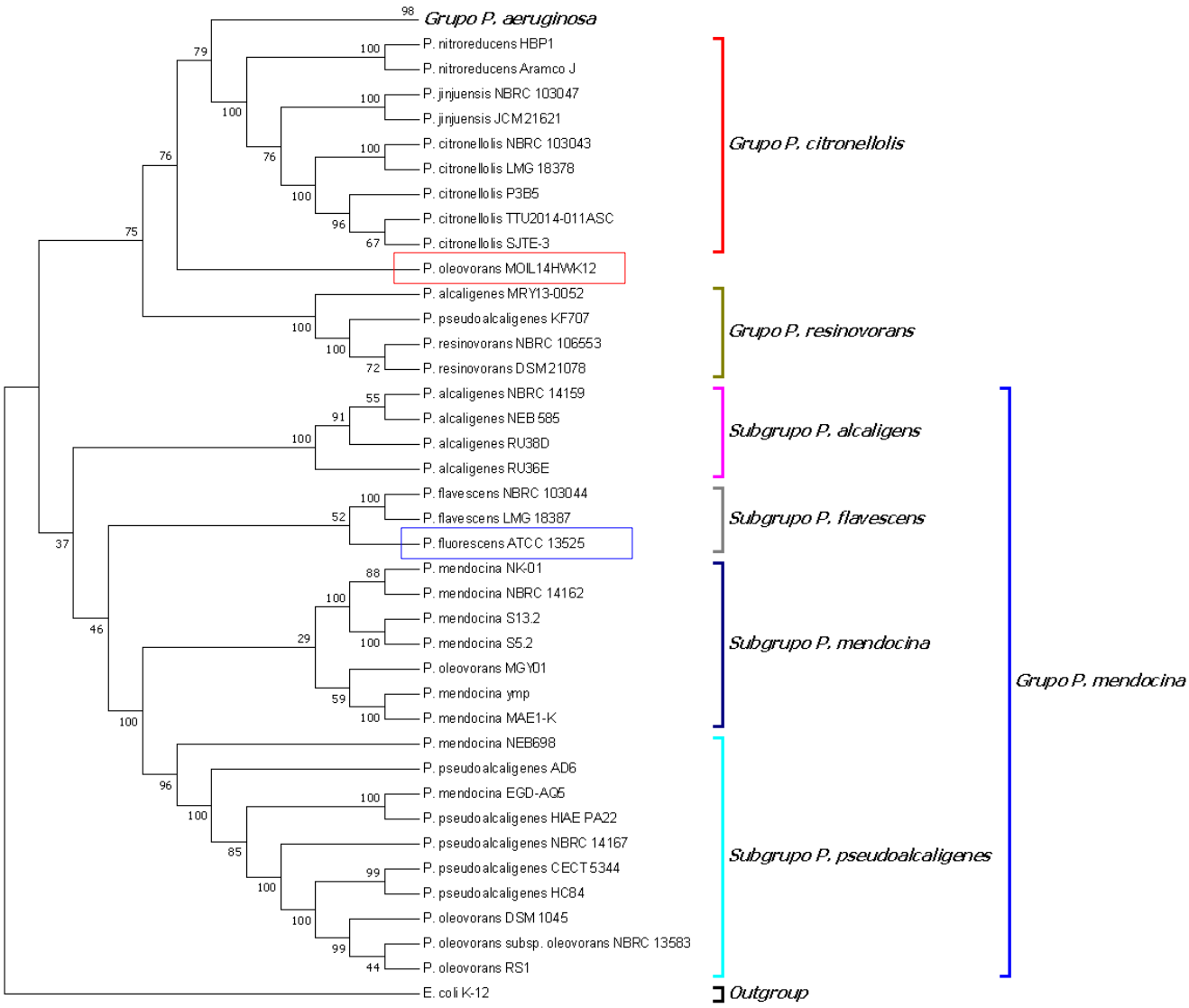

Figura 8. Árbol basado en MLSA comprimiendo el grupo de *P. aeruginosa*.

Estos grupos que se observaron en este árbol, no concuerdan con los cinco que establecieron Palleroni, et al., (17,18). De todas maneras, como se mencionó en la introducción, los análisis de MLSA no son del todo fiables, ya que al estudiar sólo cuatro genes (aunque estos sean genes contitutivos) se pierde mucha información del resto del genoma. Por todo esto, se decidió realizar un análisis de similitud mediante ANIb.

Filogenia basada en el genoma completo

El análisis realizado por MLSA es una aproximación práctica para definir los distintos grupos que se encuentran dentro del complejo de *P. aeruginosa*, gracias a la concatenación de los genes constitutivos, pero los estudios filigenómicos pueden proporcionar una mayor información ya que se incluyen también otros genes que pueden ser determinantes para diferenciar unos grupos de otros.

Como se comentó anteriormente, existen varios métodos para estudiar el genoma completo tales como el índice ANIb y GBDP, pero en este caso sólo se utilizó ANIb, puesto que como se dijo

en la introducción, los desarrolladores de GBDP no habían liberado el código todavía para poder utilizarlo en modo local. Se utilizaron como *outgroup P. fluorescens*, que se ha demostrado que es una bacteria que está lejana evolutivamente hablando (40),y *E. coli* que se emplea en muchos estudios como *outgroup* del género *Pseudomonas* (22).

A la vista de los resultados Figura 9, se establecieron 7 grupos diferenciados en base a su topología, el propio grupo de *P. aeruginosa* y otros 6 grupos que en la imagen inferior no se pueden distinguir de una manera óptima, ya que son menos numerosos. Por ello, se comprimió, como en el caso anterior, el grupo de *P. aeruginosa*, y así se definieron mejor los otros 6 grupos.

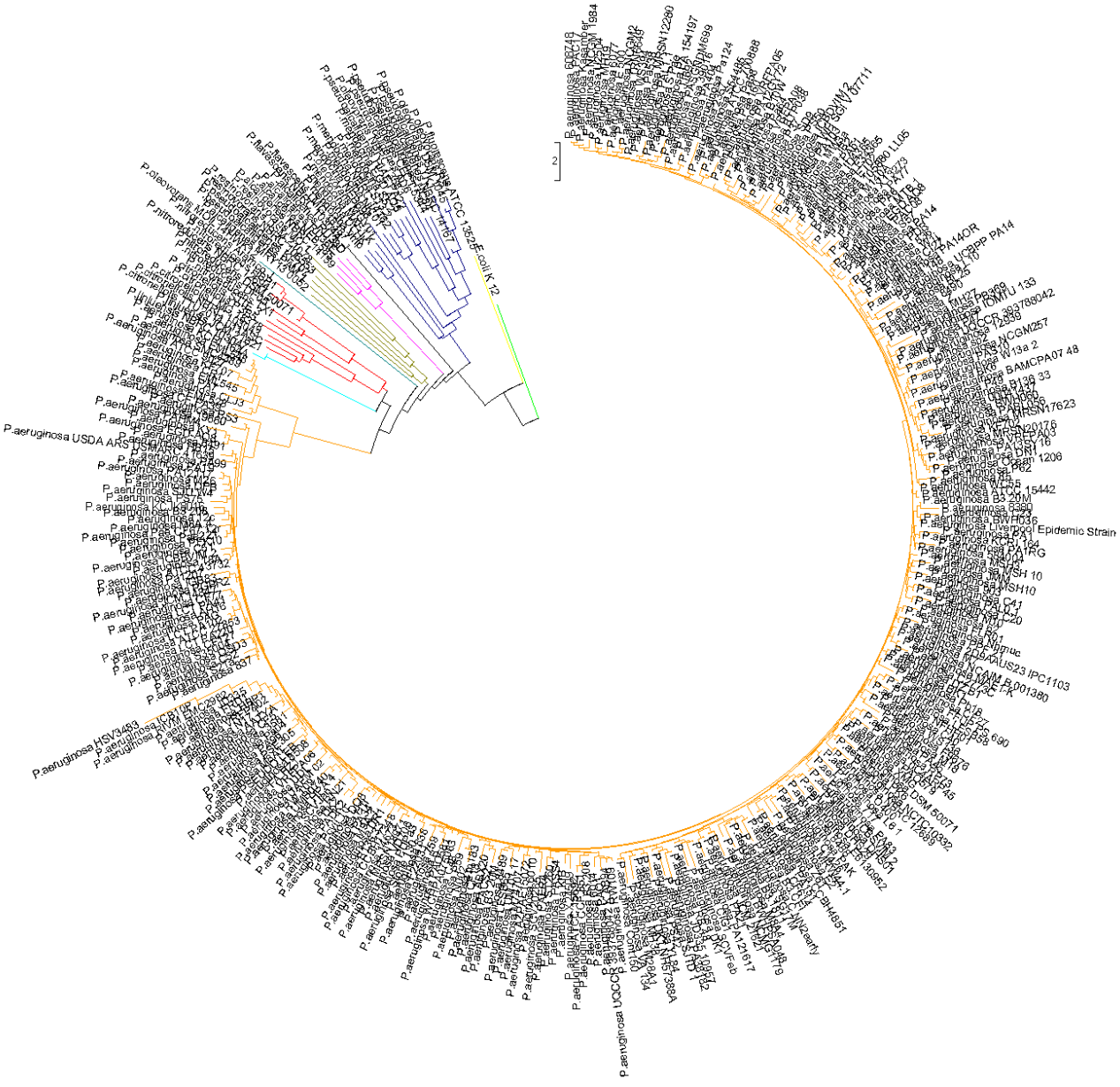

Figura 9. Árbol basado en ANIb.

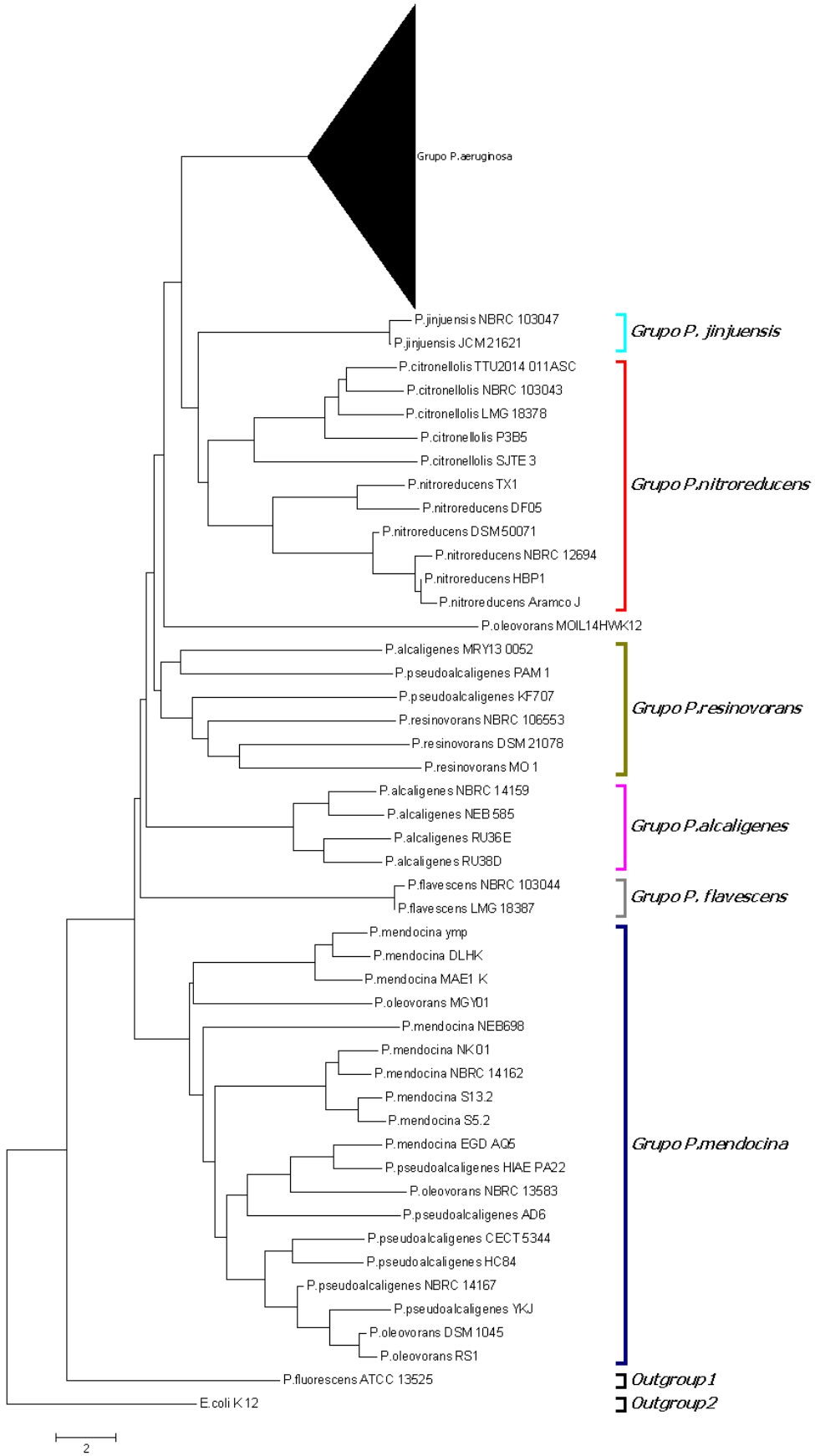

Figura 10. Árbol basado en ANIb comprimiendo el grupo de *P. aeruginosa*.

Dentro de estos 6 grupos (Figura 10), se encontraron el de *P. jinjuensis, P. nitroreducens, P. resinovorans, P. alcaligenes, P. flavescens* y *P. mendocina.* Existe una especie de *P. oleovorans* que se encuentra en mitad del árbol y que no está en el mismo grupos que las demás *P. oleovorans*, esto podría deberse a que su genoma difiere con el del resto de *oleovorans* por estar mutado o por contener contaminaciones de otras bacterias cuando realizaron la secuenciación de la misma.

Al compararlo con el árbol MLSA, se observó que existían una serie de diferencias, ya que en MLSA se encontraron 4 grupos mientras que en el árbol de ANIb fueron 6. En cuanto al grupo de *P. aeruginosa*, en los dos árboles era muy similar, ya que era el grupo más numeroso y más distal, y que por lo tanto es el que define el "complejo de *P. aeruginosa*". El grupo de *P. citronellolis* del MLSA, en el ANIb se divide en el grupo *P. jinjuensis y P. nitroreducens*. Cabe destacar que en MLSA existían algunas especies de *P. mendocina, P. pseudoalcaligenes, P. resinovorans y P. nitroreducens* que se encontraban en el grupo de *P. aeruginosa*, pero en ANIb las cepas de *P. nitroreducens* se encuentran todas juntas formando su propio grupo junto con *P. citronellolis*. En cuanto al grupo de *P. mendocina*, en MLSA se componía de 4 subgrupos (*P. alcaligenes, P. flavescens, P. mendocina y P. pseudoalcaligenes)*, mientras que en ANIb estos grupos están diferenciados propiamente como grupos individuales y no como subgrupos dentro de un grupo mayor. La única diferencia entre ellos es que en ANIb el grupo *P. mendocina* englobaría los dos subgrupos de *P. mendocina y P. pseudoalcaligenes* del MLSA. Otra de las diferencias que se observó es que en ANIb, tanto *P. fluorescens* como *E. coli* sí que están representados como *outgroup.* Aun así, este estudio filogenómico se contradice con la división que establecieron Palleroni et al. , ya que ellos defendieron que *P. aeruginosa* estaba compuesto por 5 grupos, en vez de 6 de este estudio, pero estas diferencias pueden basarse en que en estos últimos años se hayan descubierto nuevas especies de bacterias y se hayan establecido nuevos grupos para su correcta clasificación taxonómica.

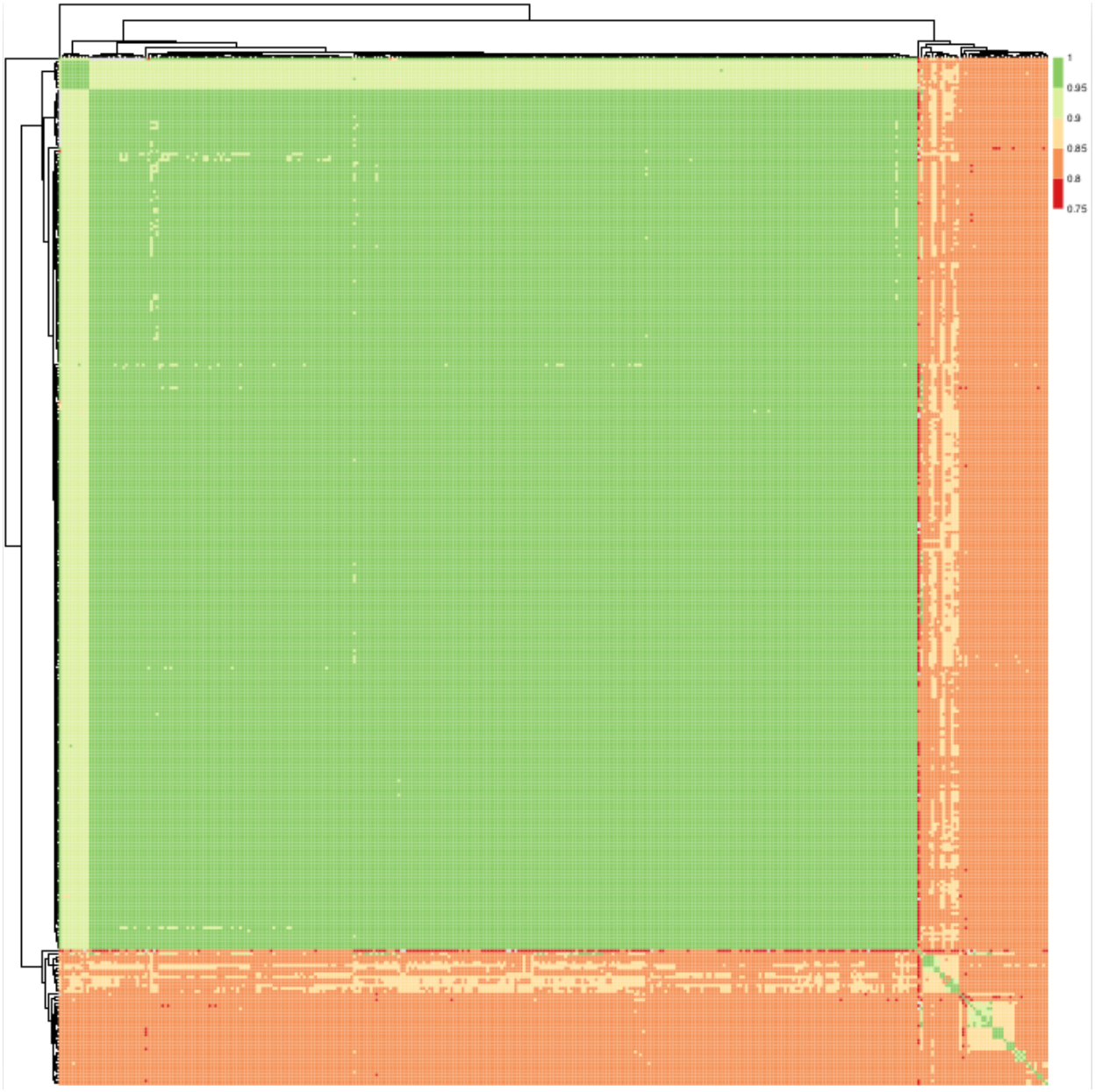

Figura 11. *Heatmap* representando la clasificación en grupos del complejo *P. aeruginosa.*

Para corroborar esta clasificación en 6 grupos, se realizó un *heatmap* o mapa de calor, a partir de la matriz de similitud de ANIb y del orden de aparición de las bacterias en el árbol de ANIb. Como se puede observar, existe un gran grupo de bacterias que tienen una gran similitud, que ocupa casi todo el *heatmap* y que se correlaciona con el grupo de P. *aeruginosa*, mientras que en la esquina derecha inferior se encontrarían los otros grupos que están definidos en el árbol ANIb. Para un mejor estudio de estos grupos inferiores, se realizó otro *heatmap* centrándolo simplemente en estos grupos.

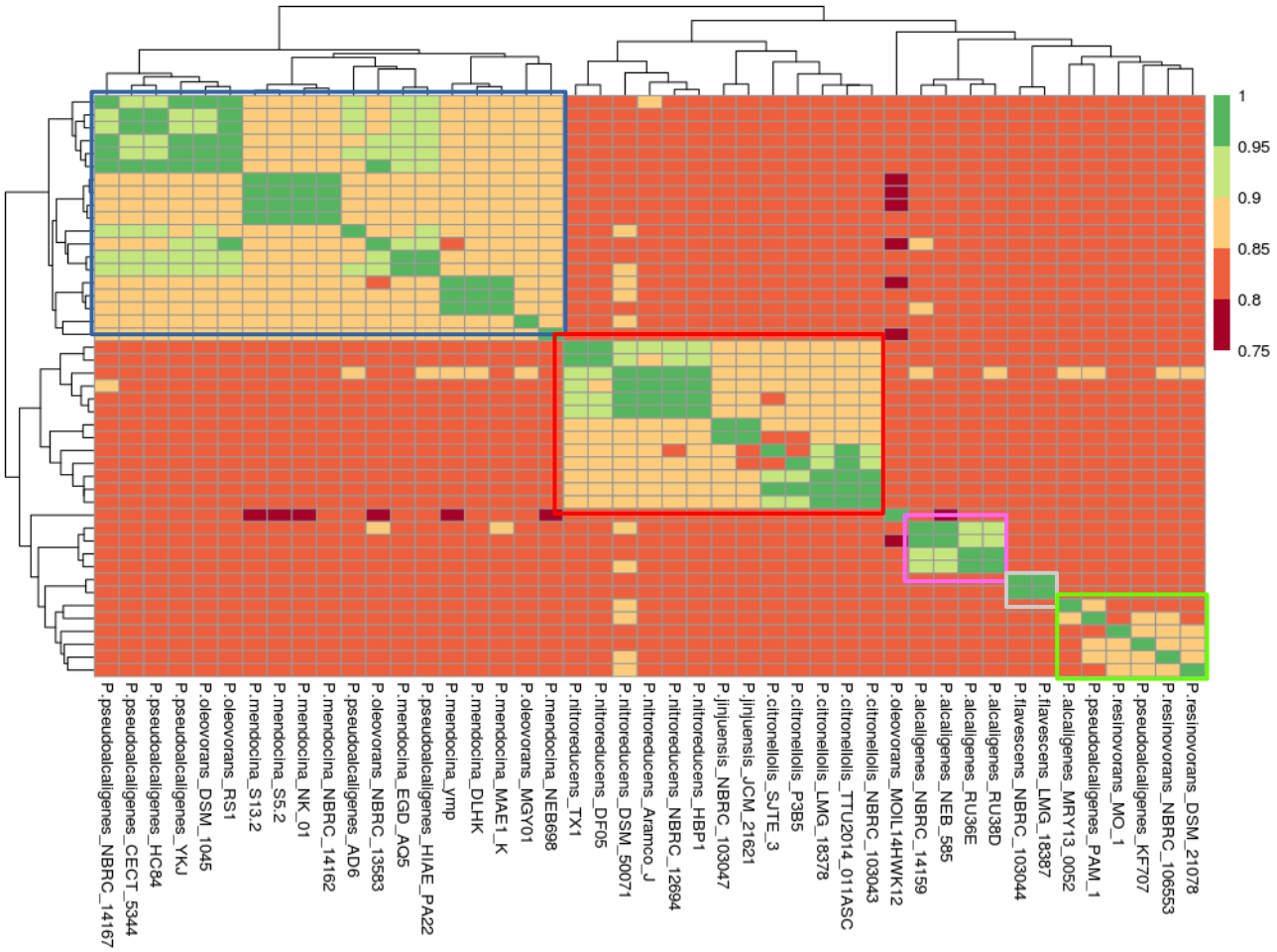

Figura 12. *Heatmap* de los 5 grupos de *P. aeruginosa* quitando el propio grupo de *P. aeruginosa.*

Como se puede observar en el *heatmap* (figura 12), existirían 5 grupos diferenciados, en azul el de *P. mendocina*, en rojo *P. nitroreducens*, en rosa *P. alcaligenes*, en gris *P. flavescens*, y en verde *P. resinovorans*. Cabe destacar, que mientras que en el árbol ANIb se estableció que *P. jinjuensis* estaba formando un grupo él sólo, en este caso se incluye dentro del grupo de *P. nitroreducens*, concordando con los que obtuvo Kwon en 2003 y Prakash et al. en 2007, quienes clasificaron *P. jinjuensis* filogenéticamente e incluyeron a este género dentro del grupo de *P. nitroreducens y P. citronellolis* (41,42). En cuanto al grupo de *P. resinovorans*, se observó que estaba muy individualizado, tal y como ocurría en el trabajo de Ashengroph et al. en 2011 (43). Por lo que respecta al resto de grupos, siguen el mismo patrón que en el árbol ANIb, al igual que *P. oleovorans MOIL14HWK12* que aparece dispersa en mitad del *heatmap*.

Por todo esto, se estableció que la clasificación de este complejo de *P. aeruginosa* era en 6 grupos: *P. aeruginosa*, *P. mendocina*, *P. nitroreducens*, *P. alcaligenes*, *P. flavescens* y *P. resinovorans*.

Estudio de las fracciones del genoma

En cuanto a las secuencias específicas de cada grupo, se obtuvieron los siguientes resultados:

Grupo	Ortogrupos específicos de grupo	Proteínas específicas de grupo
P. aeruginosa	77	2193
P. mendocina	64	837
P. nitroreducens	144	1343
P. alcaligenes	354	1421
P. flavescens	1620	1878
P. resinovorans	117	808

Figura 13. Tabla con los ortogrupos específicos de cada grupos, así como las proteínas específicas.

Como se recoge en la tabla anterior, en *P. aeruginosa* 77 ortogrupos específicos de grupo, obteniéndose 2193 proteínas específicas de grupo. En cuanto a *P. mendocina*, estos datos eran de 64 ortogrupos específicos y 837 proteínas específicas, en *P. nitroreducens* eran 144 ortogrupos y 1343 proteínas, en *P. alcaligenes* 354 ortogrupos y 1421 proteínas, en *P. flavescens* 1620 ortogrupos y 1878 proteínas, y por último en *P. resinovorans* fue de 117 ortogruos esecíficos y 808 proteínas específicas. A la vista de estos datos, se podría concluir que en *P. mendocina* y *P. resinovorans* existe un menor número de proteínas específicas de especie porque en estos grupos se incluyen un gran número de especies y muy diversas entre ellas. Mientras que *P. aeruginosa* sería el grupo con más proteínas específicas de grupo, debido a que las secuencias codificantes de estas proteínas son muy similares entre sí y no existe una gran variabilidad.

En la figura 14, se muestran los ortogrupos comunes del genoma central nuevos con respecto al número de genomas de la muestra. Se puede observar que a medida que se van incrementando el número de bacterias, el número de ortogrupos nuevos va disminuyendo.

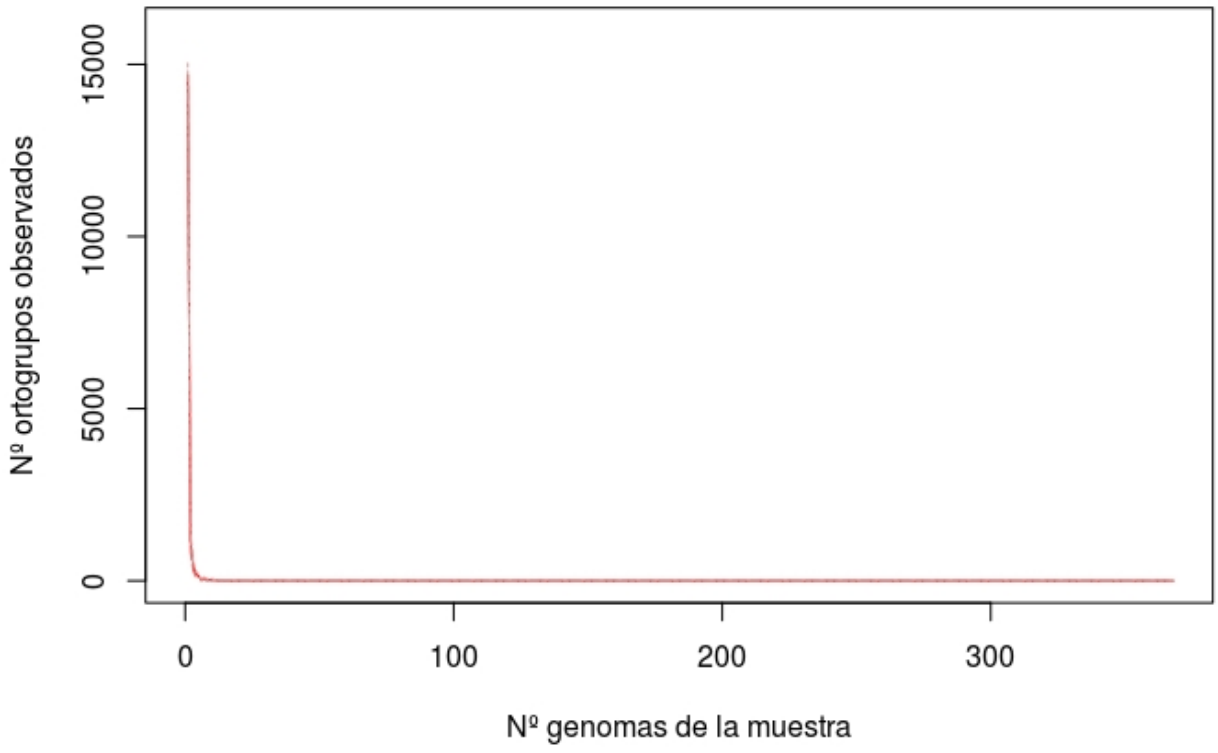

Figura 14. Número de ortogrupos nuevos obtenidos a medida que se incrementan el número de genomas de la muestra.

Estudio de las características funcionales

Al analizar las características funcionales de cada grupo, se observaron las siguientes diferencias:

- *P. aeruginosa:* se caracteriza por el metabolismo y biosíntesis de oxazol y tiazol, que son compuestos que poseen propiedades fungicidas y pesticidas. Al mismo tiempo, también sintetizan poliprenoles, los cuales son biorreguladores que actúan como moduladores en procesos celulares de plantas. Todos estos datos concuerdan con los que se expusieron en la introducción, ya que se comentó que *P. aeruginosa* estaba implicada dotaba a la planta que infectaba de ciertas propiedades antifúngicas, afectando a su vez al crecimiento de la planta (9,10).
- *P. nitroreducens:* entre los procesos biológicos que están enriquecidos en este grupo, se engloban todos aquellos que tiene que ver con la infección, ya que están enriquecidos significativamente los fenómenos de conjugación, detección de virus, empaquetamiento del ADN y la entrada en una célula huesped. Al mismo tiempo, también posee la característica de metabolizar el tolueno, lo cual permite que se pueda utilizar esta bacteria en biorremediación, tal y como describieron Yen et al. en 1991 (44).

- *P. alcaligenes:* en este caso, el proceso que está enriquecido es la producción de organofosfato, el cual es utilizado como plaguicida contra una gran variedad de insectos (45).

Conclusiones

A la vista de los resultados obtenidos en este trabajo, se puede concluir que el complejo de *P. aeruginosa* está formado por seis grupos diferenciados, entre los cuales se pueden obtener diferencias en cuanto a las características funcionales de cada uno, tales como la biorremediación, las propiedades antifúngicas y pesticidas, lo cual concordaba con los grupos establecidos en el análisis filogenómico. Al mismo tiempo, al estudiar las fracciones del genoma, se comprobó que existían unos grupos en los que existían menos proteínas ortólogas entre ellos, conluyendo que esos grupos eran más diversos filogenéticamente hablando.

Bibliografía

1. Mann EE, Wozniak DJ. Pseudomonas biofilm matrix composition and niche biology. FEMS Microbiol Rev. julio de 2012;36(4):893-916.

2. Crousilles A, Dolan SK, Brear P, Chirgadze DY, Welch M. Gluconeogenic precursor availability regulates flux through the glyoxylate shunt in Pseudomonas aeruginosa. J Biol Chem. 20 de julio de 2018;

3. Zhang R-C, Xu X-J, Chen C, Xing D-F, Shao B, Liu W-Z, et al. Interactions of functional bacteria and their contributions to the performance in integrated autotrophic and heterotrophic denitrification. Water Res. 26 de junio de 2018;143:355-66.

4. Gross H, Loper JE. Genomics of secondary metabolite production by Pseudomonas spp. Nat Prod Rep. noviembre de 2009;26(11):1408-46.

5. LPSN-List of prokaryotic names with standing in nomenclature [Internet]. Disponible en: http://www.bacterio.net/pseudomonas.html

6. Mulet M, Lalucat J, García-Valdés E. DNA sequence-based analysis of the Pseudomonas species. Environ Microbiol. junio de 2010;12(6):1513-30.

7. Gomila M, Peña A, Mulet M, Lalucat J, García-Valdés E. Phylogenomics and systematics in Pseudomonas. Front Microbiol. 2015;6:214.

8. Sulochana MB, Jayachandra SY, Kumar SKA, Dayanand A. Antifungal attributes of siderophore produced by the Pseudomonas aeruginosa JAS-25. J Basic Microbiol. mayo de 2014;54(5):418-24.

9. Hariprasad P, Chandrashekar S, Singh SB, Niranjana SR. Mechanisms of plant growth promotion and disease suppression by Pseudomonas aeruginosa strain 2apa. J Basic Microbiol. agosto de 2014;54(8):792-801.

10. Ortiz-Castro R, Pelagio-Flores R, Méndez-Bravo A, Ruiz-Herrera LF, Campos-García J, López-Bucio J. Pyocyanin, a virulence factor produced by Pseudomonas aeruginosa, alters root

development through reactive oxygen species and ethylene signaling in Arabidopsis. Mol Plant-Microbe Interact MPMI. abril de 2014;27(4):364-78.

11. Chance DL, Mawhinney TP. Carbohydrate sulfation effects on growth of Pseudomonas aeruginosa. Microbiol Read Engl. julio de 2000;146 (Pt 7):1717-25.

12. Zhang J-F, Zhu H-Y, Sun Y-W, Liu W, Huo Y-M, Liu D-J, et al. Pseudomonas aeruginosa Infection after Pancreatoduodenectomy: Risk Factors and Clinic Impacts. Surg Infect. diciembre de 2015;16(6):769-74.

13. Coetzee E, Rode H, Kahn D. Pseudomonas aeruginosa burn wound infection in a dedicated paediatric burns unit. South Afr J Surg Suid-Afr Tydskr Vir Chir. 3 de mayo de 2013;51(2):50-3.

14. Serra R, Grande R, Butrico L, Rossi A, Settimio UF, Caroleo B, et al. Chronic wound infections: the role of Pseudomonas aeruginosa and Staphylococcus aureus. Expert Rev Anti Infect Ther. mayo de 2015;13(5):605-13.

15. Migula W. System der Bakterien: Handbuch der Morphologie, Entwicklungsgeschichte und Systematik der Bakterien [Internet]. Fischer; 1897. Disponible en: https://books.google.es/books?id=yVjMjwEACAAJ

16. Stanier RY, Palleroni NJ, Doudoroff M. The Aerobic Pseudomonads a Taxonomic Study. J Gen Microbiol. 1 de mayo de 1966;43(2):159-271.

17. Palleroni NJ, Ballard RW, Ralston E, Doudoroff M. Deoxyribonucleic acid homologies among some Pseudomonas species. J Bacteriol. abril de 1972;110(1):1-11.

18. Palleroni NJ, Kunisawa R, Contopoulou R, Doudoroff M. Nucleic Acid Homologies in the Genus Pseudomonas. Int J Syst Bacteriol. 1 de octubre de 1973;23(4):333-9.

19. Santos SR, Ochman H. Identification and phylogenetic sorting of bacterial lineages with universally conserved genes and proteins. Environ Microbiol. julio de 2004;6(7):754-9.

20. Eisenberg E, Levanon EY. Human housekeeping genes, revisited. Trends Genet. octubre de 2013;29(10):569-74.

21. Adékambi T, Drancourt M, Raoult D. The rpoB gene as a tool for clinical microbiologists. Trends Microbiol. enero de 2009;17(1):37-45.

22. Harayama S, Arnold DL, Jackson RW, Yamamoto S, Kasai H, Vivian A. Phylogeny of the genus Pseudomonas: intrageneric structure reconstructed from the nucleotide sequences of gyrB and rpoD genes. Microbiology. 1 de octubre de 2000;146(10):2385-94.

23. Mulet M, Gomila M, Scotta C, Sánchez D, Lalucat J, García-Valdés E. Concordance between whole-cell matrix-assisted laser-desorption/ionization time-of-flight mass spectrometry and multilocus sequence analysis approaches in species discrimination within the genus Pseudomonas. Syst Appl Microbiol. octubre de 2012;35(7):455-64.

24. Sánchez D, Mulet M, Rodríguez AC, David Z, Lalucat J, García-Valdés E. Pseudomonas aestusnigri sp. nov., isolated from crude oil-contaminated intertidal sand samples after the Prestige oil spill. Syst Appl Microbiol. marzo de 2014;37(2):89-94.

25. Maiden MC, Bygraves JA, Feil E, Morelli G, Russell JE, Urwin R, et al. Multilocus sequence typing: a portable approach to the identification of clones within populations of pathogenic microorganisms. Proc Natl Acad Sci U S A. 17 de marzo de 1998;95(6):3140-5.

26. Rokas A, Williams BL, King N, Carroll SB. Genome-scale approaches to resolving incongruence in molecular phylogenies. Nature. octubre de 2003;425(6960):798-804.

27. Konstantinidis KT, Tiedje JM. Genomic insights that advance the species definition for prokaryotes. Proc Natl Acad Sci. 15 de febrero de 2005;102(7):2567-72.

28. Klappenbach JA, Goris J, Vandamme P, Coenye T, Konstantinidis KT, Tiedje JM. DNA–DNA hybridization values and their relationship to whole-genome sequence similarities. Int J Syst Evol Microbiol. 1 de enero de 2007;57(1):81-91.

29. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ. Basic local alignment search tool. J Mol Biol. octubre de 1990;215(3):403-10.

30. Kurtz S, Phillippy A, Delcher AL, Smoot M, Shumway M, Antonescu C, et al. Versatile and open software for comparing large genomes. Genome Biol. 30 de enero de 2004;5(2):R12.

31. Richter M, Rosselló-Móra R. Shifting the genomic gold standard for the prokaryotic species definition. Proc Natl Acad Sci. 10 de noviembre de 2009;106(45):19126-31.

32. Henz SR, Huson DH, Auch AF, Nieselt-Struwe K, Schuster SC. Whole-genome prokaryotic phylogeny. Bioinformatics. 15 de mayo de 2005;21(10):2329-35.

33. Camacho C, Coulouris G, Avagyan V, Ma N, Papadopoulos J, Bealer K, et al. BLAST+: architecture and applications. BMC Bioinformatics. 2009;10(1):421.

34. Kent WJ. BLAT---The BLAST-Like Alignment Tool. Genome Res. 20 de marzo de 2002;12(4):656-64.

35. Meier-Kolthoff JP, Hahnke RL, Petersen J, Scheuner C, Michael V, Fiebig A, et al. Complete genome sequence of DSM 30083T, the type strain (U5/41T) of Escherichia coli, and a proposal for delineating subspecies in microbial taxonomy. Stand Genomic Sci. 2014;9(1):2.

36. Sievers F, Wilm A, Dineen D, Gibson TJ, Karplus K, Li W, et al. Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega. Mol Syst Biol. 16 de abril de 2014;7(1):539-539.

37. Tamura K, Stecher G, Peterson D, Filipski A, Kumar S. MEGA6: Molecular Evolutionary Genetics Analysis Version 6.0. Mol Biol Evol. diciembre de 2013;30(12):2725-9.

38. Rstudio Team RT. RStudio: Integrated Development for R [Internet]. Boston, MA: RStudio, Inc.; 2015. Disponible en: http://www.rstudio.com/

39. Emms DM, Kelly S. OrthoFinder: solving fundamental biases in whole genome comparisons dramatically improves orthogroup inference accuracy. Genome Biol [Internet]. diciembre de 2015 [citado 28 de agosto de 2018];16(1). Disponible en: http://genomebiology.com/2015/16/1/157

40. Garrido-Sanz D, Meier-Kolthoff JP, Göker M, Martín M, Rivilla R, Redondo-Nieto M. Genomic and Genetic Diversity within the Pseudomonas fluorescens Complex. Vinatzer BA, editor. PLOS ONE. 25 de febrero de 2016;11(2):e0150183.

41. Prakash O, Kumari K, Lal R. Pseudomonas delhiensis sp. nov., from a fly ash dumping site of a thermal power plant. Int J Syst Evol Microbiol. 1 de marzo de 2007;57(3):527-31.

42. Kwon SW. Pseudomonas koreensis sp. nov., Pseudomonas umsongensis sp. nov. and Pseudomonas jinjuensis sp. nov., novel species from farm soils in Korea. Int J Syst Evol Microbiol. 1 de enero de 2003;53:21-7.

43. Ashengroph M, Nahvi I, Zarkesh-Esfahani H, Momenbeik F. Pseudomonas resinovorans SPR1, a newly isolated strain with potential of transforming eugenol to vanillin and vanillic acid. New Biotechnol. octubre de 2011;28(6):656-64.

44. Yen KM, Karl MR, Blatt LM, Simon MJ, Winter RB, Fausset PR, et al. Cloning and characterization of a Pseudomonas mendocina KR1 gene cluster encoding toluene-4-monooxygenase. J Bacteriol. septiembre de 1991;173(17):5315-27.

45. Duneau D, Sun H, Revah J, San Miguel K, Kunerth HD, Caldas IV, et al. Signatures of Insecticide Selection in the Genome of Drosophila melanogaster. G3 GenesGenomesGenetics [Internet]. 6 de septiembre de 2018; Disponible en: http://www.g3journal.org/content/early/2018/09/06/g3.118.200537.abstract

Anexos

Scripts de la memoria del TFM:

ftp.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

#cargar las librerías que se van a utilizar
import os
import sys
import re
import subprocess

#abrir el archivo de la base de datos donde se encuentran todos los datos de los
genomas,así como los enlaces de descarga de los genomas.
archivo= open('genomas_pseudomonas.txt','r')
for linea in archivo:

#omitir la línea que empieza por # y pasar a la siguiente
      if linea.startswith("#"):
            continue

#Separar la línea por el tabulador y guardar los campos 0 y 18. Nombrar a la
carpeta según la carpeta que contiene los archivos ftp de cada genoma.
      else:
            line=linea.split("\t")
            identificador= line[18]
```

```
            esp= line[0]
            especie= re.sub(" ", "_", esp)
            carpeta= os.path.basename(identificador)

#Escribir el comando que se utilizó para descargar los datos y ejecutarlo con
subprocess.call
            command= "wget "+identificador+"/*.gz -P ./"+carpeta+"/"
            subprocess.call(command, shell=True)
```

cambiar_nombre_id.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

#cargar las librerías que se van a utilizar
import os
import sys
import re
import subprocess
import distutils.dir_util
from Bio import SeqIO
from BCBio import GFF

#obtener el directorio raíz
directorio=os.getcwd()

#ordenar los archivos que se encuentran en el directorio
file_list = sorted(os.listdir(directorio))

#crear las carpetas MLSA y GFF, donde se incluirán los archivos que se generarán
distutils.dir_util.mkpath(directorio+"/MLSA")
distutils.dir_util.mkpath(directorio+"/GFF")

#abrir el archivo con todos los datos de los genomas de Pseudomonas
gen_pse=open('./genomas_pseudomonas.csv','r')


#recorrer el fichero línea a línea y si no empieza por # continuar.
for line in gen_pse:
      if not line.startswith('#'):
            linea=line.split('\t')

#guardar el el identificador del genoma, sustituir la A por F, y quitar los
espacios en blanco
            assembly=linea[7]
            gcf=re.sub('A','F',assembly)
            id_strain=re.sub(' ','',gcf)

#recorrer los archivos de la lista de archivos,comprobar que es un directorio, y
si lo es, meterse en la carpeta raíz y obtener el nuevo directorio
            for archivo_ass in file_list:
                  patron=archivo_ass
                  comprob=os.path.isdir(archivo_ass)
                  if comprob==True:
                        ruta_arch=os.path.abspath(archivo_ass)
                        if id_strain in archivo_ass:
```

```
                              os.chdir(directorio+'/'+patron)
                              nd=os.getcwd()

#ordenar este nuevo directorio y recorrer la lista de archivos
                              list_gz=sorted(os.listdir(nd))
                              for fi_gz in list_gz:

#reemplazar el nombre del archivo por el identificador de la especie
                                    os.rename(fi_gz, fi_gz.replace (archivo_ass,
id_strain))

#cambiar el directorio al directorio incial y renombrar el nombre de la carpeta
                              os.chdir(directorio)
                              os.rename(ruta_arch,id_strain)
```

seq.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

#cargar las librerías que se van a utilizar
import os
import sys
import re
import subprocess
import distutils.dir_util
from Bio import SeqIO
from BCBio import GFF

#recorrer la lista de archivos del directorio inicial, y si es un directorio
acceder a la carpeta y ordenar los ficheros
for archivo in file_list:
      comprob=os.path.isdir(archivo)
      if comprob==True:
            print 'directorio',archivo
            os.chdir(directorio+'/'+archivo)
            new_dir=os.getcwd()
            new_arch=sorted(os.listdir(new_dir))

#recorrer la nueva lista de archivos y si el archivo termina en gff.gz copiarlo
a la carpeta GFF y volver al directorio inicial
            for arch in new_arch:
                  if arch.endswith('gff.gz'):
                        ruta_arch_gz=os.path.abspath(arch)
                        os.system("cp {} {}/GFF".format (ruta_arch_gz,
directorio))
            os.chdir(directorio)

#al terminar de copiar los gff.gz a la carpeta GFF, acceder a esta carpeta y
ordenar los ficheros
os.chdir(directorio+'/GFF')
os.system("gunzip *.gz")
file_list2 = sorted(os.listdir(directorio+'/GFF'))

#crear los archivos que incluirán las secuencias de los genes gryB,rpoB,rpoD y
16S e incluir una cabecera para saber a lo que corresponde cada columna.
gyrB_arch_mlsa=open('./bbdd_mlsa_gyrB.txt','w')
rpoB_arch_mlsa=open('./bbdd_mlsa_rpoB.txt','w')
rpoD_arch_mlsa=open('./bbdd_mlsa_rpoD.txt','w')
```

```
diec6_arch_mlsa=open('./bbdd_mlsa_16S.txt','w')
header='\t'.join(['##Nombre_Archivo','ID','Start','End','Strand','Ruta_archivo_g
ff','Secuencia', 'Gen'])
gyrB_arch_mlsa.write(header+'\n')
rpoB_arch_mlsa.write(header+'\n')
rpoD_arch_mlsa.write(header+'\n')
diec6_arch_mlsa.write(header+'\n')

#recorrer la lista de archivos de la carpeta GFF y abrir cada archivo .gff
for archiv in file_list2:
      if archiv.endswith('.gff'):
            ruta_arch_gff=os.path.abspath(archiv)
            nombre_archivo=os.path.basename(ruta_arch_gff)
            file_gff=open(archiv,'r')

#recorrer cada archivo y si la línea coincide con el siguiente patrón, guardar
en distintas variables los campos necesarios para nuestra base de datos
            for line in file_gff:
                  coinc=re.match
("(.*)\\t(.*)\\t(.*)\\t(.*)\\t(.*)\\t(.*)\\t(.*)\\t(.*)\n", line)
                  if coinc is not None:
                        id_gene,gene,start,end,strand,descrip = coinc.group(1),
coinc.group(3), coinc.group(4), coinc.group(5), coinc.group(7), coinc.group(9)

#encontrar la secuencia parcial del gen gyrB
                        if 'gyrB' in descrip and gene=='gene':
                              linea_gyrB="\t".join ([nombre_archivo, id_gene,
start, end, strand, ruta_arch_gff,'gyrB'])
                              gyrB_arch_mlsa.write(linea_gyrB+'\t\n')

#y si no, encontrar la secuencia parcial del gen rpoD
                        else:
                              if'rpoD' in descrip and gene=='gene':
                                    linea_rpoD="\t".join ([nombre_archivo,
id_gene, start, end, strand, ruta_arch_gff, 'rpoD'])
                                    rpoD_arch_mlsa.write(linea_rpoD+'\t\n')

#y si no , me encuentra la secuencia parcial del gen rpoB
                              else:
                                    if 'rpoB' in descrip and gene=='gene':
                                          linea_rpoB="\t".join ([nombre_archivo,
id_gene, start, end, strand, ruta_arch_gff, 'rpoB'])
                                          rpoB_arch_mlsa.write
(linea_rpoB+'\t\n')

#y si no, encontrar la secuencia parcial del rRNA 16S, utilizar break para que
en cuanto encuentre una coincidencia la guarde y pase al siguiente genoma.
                                    else:
                                          if '16S' in descrip and gene=='rRNA':

                                                linea_16S="\t".join
([nombre_archivo, id_gene, start, end, strand, ruta_arch_gff, '16S'])
                                                diec6_arch_mlsa.write
(linea_16S+'\t\n')
                                                break
#cerrar el archivo .gff
            file_gff.close()
```

```
#cerrar las bases de datos de las secuencias de los genes
gyrB_arch_mlsa.close()
rpoD_arch_mlsa.close()
rpoB_arch_mlsa.close()
diec6_arch_mlsa.close()

#copiar los archivos anteriores a la carpeta MLSA
os.system("cp *.txt {}/MLSA".format(directorio))
```

prueba_seqIO.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

#cargar las librerías
import os
import sys
import re
import subprocess
import distutils.dir_util
from Bio import SeqIO
from Bio.SeqRecord import SeqRecord
from Bio.Seq import Seq

#obtener el directorio de trabajo, ordenar los archivos que contiene y crear las
carpetas donde se guardarán los archivos.
directorio=os.getcwd()
file_list = sorted(os.listdir(directorio))
distutils.dir_util.mkpath(directorio+"/FNA")
distutils.dir_util.mkpath(directorio+"/BLAST")

#generar las variables y los nombres de los genes
fasta_gyr=''
fasta_rpoB=''
fasta_rpoD=''
nombre_archivo=''
gyrB='gyrB'
rpoB='rpoB'
rpoD='rpoD'
diec6='16S'

#definir la función de búsqueda en BLASTN
def blast_search(gen):
      identificador=[]

#abrir el archivo de la bbdd de las secuencias de los genes ya encontrados en la
anotación
      arch_mlsa=open(directorio+'/MLSA/bbdd_mlsa_'+gen+'.txt','r')

#establecer el archivo donde se encuentra la secuencia tipo de los genes gyrB,
rpoB, rpoD y 16S
      arch_mlsa_fasta=directorio+'/'+gen+'_type_fasta'

#recorrer el fichero, guardar el identificador del archivo y guardarlo en una
lista
      for line_fasta in arch_mlsa:
            linea=line_fasta.split('\t')
```

```
            ident=linea[0]
            identificador.append(ident)
      arch_mlsa.close()

#recorrer los archivos de la carpeta FNA y si el archivo termina en genomic.fna,
guardar con el nombre del archivo
      for archivo_fna in file_list2:
            if archivo_fna.endswith('genomic.fna'):
                  arc=re.sub("_genomic.fna","",archivo_fna)

#recorrer la lista de identificadores y si se encuentra el archivo entre los
identificadores, poner un flag True y si no False.
                  for i in identificador:
                        if arc in str(identificador):
                              coincidencia=True
                        if arc not in str(identificador):
                              coincidencia=False

#si no encuentra ese identificador, hacer BLASTN para encontrar la secuencia de
ese gen, en el que están distintos parámetros de blastn: query (la secuencia
tipo de cada uno de los genes que se quería buscar), db (el archivo de la base
de datos en el que se buscó esta secuencia) y outfmt (es el formato en el que se
exportaron los datos y los campos necesarios con la información del hit de
blastn).
                  if coincidencia==False:
                        os.system("blastn -query {} -db {} -outfmt '7 qseqid
sseqid pident length mismatch gapopen qstart qend sstart send evalue bitscore
sstrand qcovs qcovhsp' -out {}_{}_7".format(arch_mlsa_fasta,arc,arc,gen))
                        os.system("blastn -query {} -db {} -outfmt '6 qseqid
sseqid pident length mismatch gapopen qstart qend sstart send evalue bitscore
sstrand qcovs qcovhsp' -out {}_{}_6".format(arch_mlsa_fasta,arc,arc,gen))
                        os.system("cp {}_{}_6
{}/BLAST".format(arc,gen,directorio))
      identificador=[]

#definir la función para extraer la secuencia de los genes, pero como estas
secuencias podían provenir de la anotación genómica o de la búsqueda por BLASTN,
se realizó en dos partes, primero las que provenían de la anotación genómica:
def blast_mlsa(gen):
      list_blast=sorted(os.listdir(directorio+'/BLAST/'))

#recorrer los archivos de la carpeta FNA y guardar el identificador
      for archivo_fna in file_list2:
            if archivo_fna.endswith('genomic.fna'):
                  arc=re.sub("_genomic.fna","",archivo_fna)

#abrir la bbdd de las secuencias obtenidas de la anotación
                  arch_mlsa=open(directorio+'/MLSA/bbdd_mlsa_'+gen+'.txt','r')

#recorrer el fichero y guardar los campos del identificador,el inicio de la
secuencia, el final y la hebra en que está
                  for line in arch_mlsa:
                        if not line.startswith('#'):
                              linea=line.split("\t")
                              arch=linea[0]
                              iden, start, end, strand= linea[1], int(linea[2]),
int(linea[3]), linea[4]
```

```
#si el identificador del archivo es igual al nombre de archivo que está en la
línea del archivo mlsa
                              if arc in arch:

#codificar la hebra, ya que para blastdbcmd se necesita que sea plus o minus
                                    if strand=='+':
                                          strand='plus'
                                    if strand=='-':
                                          strand='minus'

#escribir el comando que se va a utilizar para extraer la secuencia del gen, en
el cual se observa que se utilizó el comando blastdbcmd (que extrae la secuencia
genómica parcial de cada gen) seguido de las opciones db (en este caso fue el
archivo en el que se encontró la secuencia), entry (el identificador de la
secuencia genómica), range (el rango en el que estaba incluida la secuencia),
strand (la hebra en la que se encontraba), outfmt (el formato en el que se sacó
la secuencia, que en este caso es formato fasta) y out (nombre del archivo en el
que se extrajo la secuencia).
                                    os.system('blastdbcmd -db {} -entry {}
-range {}-{} -strand {} -outfmt %f -out {}_out_{}'.format (arc, iden, start,
end, strand, arc, gen))

# y en segundo las que se obtuvieron de BLASTN, con lo cual se recorrió la lista
de archivos de la carpeta BLAST
                  for archivo_blast in list_blast:
                        arch_blast=directorio+"/BLAST/"+arc+'_'+gen+'_6'

#si exsite el archivo, abrir el fichero y si la primera línea no está vacía,
guardamos con el strand
                        if os.path.exists(arch_blast):
                              with open(arch_blast) as f:
                                    first_line = f.readline()
                                    if first_line!='':
                                          li_blast=first_line.split("\t")
                                          strand=li_blast[12]

#si la hebra es +, el empiece y el final de la secuencia corresponden con los
que teníamos en el archivo
                                          if strand=='plus':
                                                ref, start, end, strand=
li_blast[1], li_blast[8], li_blast[9], li_blast[12]

#pero si es -, cambiar el orden del principio y fin, porque blastdbcmd no admite
que el principio sea mayor que el final
                                          if strand=='minus':
                                                ref, start, end, strand=
li_blast[1], li_blast[9], li_blast[8], li_blast[12]

#guardar el identificador de la proteina y crear el comando para extraer la
secuencia
                                          ident=re.sub('ref\|','',ref)
                                          id=re.sub('\|','',ident)
                                          os.system('blastdbcmd -db {} -entry {}
-range {}-{} -strand {} -outfmt %f -out {}_out_{}'.format(arc, id, start, end,
strand, arc, gen))

#recorrer la lista de archivos del directorio de trabajo, y si el archivo es un
directorio, acceder en esa carpeta
```

```
for archivo in file_list:
      comprob=os.path.isdir(archivo)
      if comprob==True:
            os.chdir(directorio+'/'+archivo)
            new_dir=os.getcwd()

#ordenar los ficheros de ese directorio y si coincide con genomic.fna.gz,
copiarlo a la carpeta FNA y volver al directorio inicial
            new_arch=sorted(os.listdir(new_dir))
            for arch in new_arch:
                  coinc= re.match((str(archivo)+'_genomic.fna.gz'),arch)
                  if coinc is not None:
                        ruta_arch_gz=os.path.abspath(arch)
                        os.system("cp {} {}/FNA".format(ruta_arch_gz,
directorio))
            os.chdir(directorio)

#cambiar al directorio FNA, descomprimir los archivos y ordenarlos
os.chdir(directorio+'/FNA')
os.system("gunzip *.gz")
file_list2=sorted(os.listdir(directorio+'/FNA'))

#por cada archivo de esta carpeta, crear la base de datos correspondiente a cada
uno de ellos
for archivo_fna in file_list2:
      if archivo_fna.endswith('genomic.fna'):
            arc=re.sub("_genomic.fna","",archivo_fna)
            os.system("makeblastdb -in {} -dbtype nucl -parse_seqids -out
{}".format (archivo_fna,arc))

#ejecutar la función de búsqueda de las secuencias con BLASTN
blast_search(gyrB)
blast_search(rpoB)
blast_search(rpoD)
blast_search(diec6)

#extraer la secuencia de cada gen mediante la función blast_mlsa
blast_mlsa(gyrB)
blast_mlsa(rpoB)
blast_mlsa(rpoD)
blast_mlsa(diec6)

#cuando se terminaron de extraer las secuencias, se definió la función para
cambiar el nombre del identificador de la proteína por el de la bacteria, ya que
en el caso de que los identificadores de la secuencia fueran diferentes, por
ejemplo al encontrarse en distintos contigs, el script diseñado para concatenar
las secuencias se encontraría con problemas
def write_fasta(gen):

#ordenar el directorio en el que estan los archivos que contienen las secuencias
      file_list_out=sorted(os.listdir(directorio+'/FNA'))
      for archivo_out in file_list_out:

#si el archivo termina con _out_y el gen, guardar el identificador del archivo
(bacteria)
            if archivo_out.endswith('_out_'+gen):
                  arc_out=re.sub('_out_'+gen,'',archivo_out)
```

```
#crear otro archivo con el mismo nombre que el anterior pero con _mod
(modificado)
                  filew = open(archivo_out+'_mod', "w")

#abrir el fichero con la secuencia y parsearlo, guardando la descripcion y el
identificador
                  with open (archivo_out, 'rU') as handle:
                        for record in SeqIO.parse(handle, "fasta"):
                              descript=record.description
                              iden_seq=record.id

#escribir el nuevo .fasta con el identificador de la bacteria, la descripcion y
la secuencia fasta
                              my_seqs = (SeqRecord (Seq (str (record.seq)),
id=arc_out, description=descript))
                              SeqIO.write(my_seqs, filew, 'fasta')

#definir la funcion para concatenar en un solo fasta, las secuencias de cada
gen, es decir, en un solo fasta las secuencias del gyrB, en otro del rpoB, etc.
def concat_fasta(gen):

#abrir el archivo en el que se van a incluir todos los fasta
      fasta_mlsa=open(directorio+'/MLSA/fasta_mlsa_'+gen+'.txt','a')

#ordenar el fichero en el que estan los archivos modificados de la anterior
función
      file_list_fna=sorted(os.listdir(directorio+'/FNA'))
      for archivo_mod in file_list_fna:

#si el archivo termina por mod, abrir el archivo y leerlo
            if archivo_mod.endswith('_out_'+gen+'_mod'):
                  arch_blast_fasta=open(archivo_mod,'r')

#por cada línea de esta archivo, escribir esa misma línea en el archivo conjunto
de todos los fasta de un mismo gen
                  for line_blast in arch_blast_fasta:
                        fasta_mlsa.write(line_blast)

#ejecutar la función de cambiar los identificadores
write_fasta(gyrB)
write_fasta(rpoB)
write_fasta(rpoD)
write_fasta(diec6)

#ejecutar la función de concatenar los fasta
concat_fasta(gyrB)
concat_fasta(rpoB)
concat_fasta(rpoD)
concat_fasta(diec6)
```

concat_MLSA.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

#cargar las librerias
import os
```

```
import sys
import re
import subprocess
import distutils.dir_util
from Bio import SeqIO
from Bio.SeqRecord import SeqRecord
from Bio.Seq import Seq

#crear las listas en las que se guardarán las secuencias, identificadores y
descripción de especie
lista_gyrB=[]
lista_rpoB=[]
lista_rpoD=[]
lista_16S=[]

#abrir cada archivo alineado de cada gen, recorrerlo y guardar cada registro en
una lista
with open ('./alin_gyrB', 'rU') as handle:
      for record_gyrB in SeqIO.parse(handle, "fasta"):
            my_seqs_gyrB = (SeqRecord(Seq(str(record_gyrB.seq)), id=
record_gyrB.id, description=record_gyrB.description))
            lista_gyrB.append(my_seqs_gyrB)

with open ('./alin_rpoB', 'rU') as handle:
      for record_rpoB in SeqIO.parse(handle, "fasta"):
            my_seqs_rpoB = (SeqRecord(Seq(str(record_rpoB.seq)), id=
record_rpoB.id, description=record_rpoB.description))
            lista_rpoB.append(my_seqs_rpoB)

with open ('./alin_rpoD', 'rU') as handle:
      for record_rpoD in SeqIO.parse(handle, "fasta"):
            my_seqs_rpoD = (SeqRecord(Seq(str(record_rpoD.seq)), id=
record_rpoD.id, description=record_rpoD.description))
            lista_rpoD.append(my_seqs_rpoD)

with open ('./alin_16S', 'rU') as handle:
      for record_16S in SeqIO.parse(handle, "fasta"):
            my_seqs_16S = (SeqRecord(Seq(str(record_16S.seq)), id=
record_16S.id, description=record_16S.description))
            lista_16S.append(my_seqs_16S)

#recorrer las cuatro listas de los genes y cuando los identificadores de la
bacteria son iguales, concatenar las secuencias de los 4 genes y guardar su
identificador y la descripción de la bacteria
for i in range(len(lista_gyrB)):
      for j in range(len(lista_rpoB)):
            for l in range(len(lista_rpoD)):
                  for m in range(len(lista_16S)):
                        if lista_16S[m].id == lista_rpoD[l].id ==
lista_rpoB[j].id ==lista_gyrB[i].id:
                  arch_mlsa=open ("./MLSA/"+lista_gyrB[i].id+"_mlsa_concat.txt",
"w")
                              my_seqs= SeqRecord
(lista_gyrB[i].seq+lista_rpoB[j].seq+lista_rpoD[l].seq+lista_16S[m].seq,
id=lista_gyrB[i].id, description=lista_gyrB[i].description)
                              SeqIO.write(my_seqs, arch_mlsa, 'fasta')
```

```
#como el nombre de la bacteria era demasiado largo, se procedió a cambiar el
nombre por el nombre corto de la misma, para ello, se abrió la base de datos en
la que están los identificadores de la especie bacteriana con el nombre de la
especie. También se creó el archivo en el que se van a guardar los nuevos
nombres cortos bacterianos
nombres_cepa=open('/home/bioinfo/Escritorio/bbdd_nombre_cepa.txt','r')
mlsa_fasta_mod=open('/home/bioinfo/Escritorio/Sara_Perez/PEN/MLSA/mlsa_concat_al
l_mod.fasta','w')
for line in nombres_cepa:
      linea_n=line.strip('\n')
      linea=linea_n.split('\t')
      id_especie,name_esp=linea[0],linea[1]

#cuando el identifciador de la bacteria es igual al que se encuentra en el
fasta,guardar el nombre corto de la bacteria junto con la secuencia en el fasta
modificado
      with open
('/home/bioinfo/Escritorio/Sara_Perez/PEN/MLSA/mlsa_concat_all.fasta', 'rU') as
handle:
            for record in SeqIO.parse(handle, "fasta"):
                  if id_especie in record.id:
                        seq_fasta_mod= SeqRecord (record.seq, id=name_esp,
description='')
                        print id_especie,name_esp
                        SeqIO.write(seq_fasta_mod, mlsa_fasta_mod, 'fasta')
```

matriz_anib.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

#cargar las librerias
import os
import sys
import re
import subprocess
import distutils.dir_util
import numpy as np
import pandas as pd

#abrir el archivo de la matriz de similitud sin los encabezamientos de las
columnas y las filas, convertir cada celda en un número decimal y realizar la
conversión de matriz de similitud en distancia.
data_anib= open('/home/bioinfo/Escritorio/perc_ids_7_modif.txt', 'r')
data_1_anib=open('/home/bioinfo/Escritorio/distance_matrix.txt','w')
for line in data_anib:
      linea=line.split('\t')
      for k in linea:
            k=float(k)
            k=100-(k*100)
            data_1_anib.write(str(k)+'\t')
      data_1_anib.write('\n')
data_1_anib.close()
```

crear_meg.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
```

```
#cargar las librerias
import os
import sys
import re
import subprocess
import distutils.dir_util
import numpy as np
import pandas as pd

#abrir la matriz de distancias como una matriz de numpy obteniendo solamente la
mitad inferior de la matriz de distancias y guardarla en un nuevo fichero
data_1_anib_str=np.genfromtxt('/home/bioinfo/Escritorio/distance_matrix.txt',
delimiter='\t')
matrix_lower=np.tril(data_1_anib_str,k=-1)
np.savetxt('/home/bioinfo/Escritorio/distance_matriz_tri.txt',matrix_lower,
delimiter='\t',fmt='%4.10f')

#ordenar la carpeta donde se encuentran los genomas de las bacterias y crear el
archivo .meg
file_list=sorted(os.listdir('/media/bioinfo/Sara/ANI_finales/FNA_buenos'))
arch_meg=open('/home/bioinfo/Escritorio/prueba.meg','w')
iden=[]

#por cada archivo de genoma de bacteria, quitar la extension _genomic.fna y
guardar solo el identificador de la bacteria
for arch_fna in file_list:
      if 'GCF' in arch_fna:
            arch=re.sub('_genomic.fna','',arch_fna)
            iden.append(arch)

#escribir las cabeceras que necesita el programa MEGA para funcionar
arch_meg.write('#mega\n')
arch_meg.write('!Title: Concatenated Files;\n')
arch_meg.write('!Format DataType=Distance DataFormat=LowerLeft NTaxa=356;\n')

#crear un contador para que nos cree un índice con los identificadores de las
bacterias
c=0
for i in iden:
      c+=1
arch_bbdd_nombre_cepa=open('/home/bioinfo/Escritorio/bbdd_nombre_cepa.txt','r')
      for line in arch_bbdd_nombre_cepa:
            linea=line.split('\t')
            id_esp,name_esp=linea[0],linea[1]
            if i in id_esp:
                  nuevo_name=name_esp

#cada vez que encuentre un identificador, suma 1 en el contador y guarda la
línea con la estructura [1] #GCF_000005845.2
                  inex=''.join(['[',str(c),']',' '])
                  linea_meg=''.join([inex,'#',name_esp])
                  arch_meg.write(linea_meg)
arch_meg.write('\n')

#mientras que se esté dentro del range de las 356 bacterias, crear la cabecera
de la matriz de distancias con los índices de cada genoma.
for l in range(c):
```

```
      if l==0:
            line_0='[  '+'     1'
            arch_meg.write(line_0)
      else:
            l+=1
            line='     '+(str(l))
            arch_meg.write(line)
      if l==356:
            line_f=' ]'
            arch_meg.write(line_f+'\n')

#guardar el índice de cada fila (índice de la bacteria) y cada valor de la
matriz de distancias
arch_matrix=open('/home/bioinfo/Escritorio/distance_matriz_tri.txt','r')
contador=0
for linea in arch_matrix:
      contador+=1
      linea_n=linea.rstrip('\n')
      line=linea_n.split('\t')
      linea_meg='['+str(contador)+']  '
      arch_meg.write(linea_meg)
      for k in line:
            k=str(k)
            if k=='0.0000000000':
                  linea_meg_nueva='      '
                  arch_meg.write(linea_meg_nueva)
            else:
                  linea_meg_nueva=str(k)+' '
                  arch_meg.write(linea_meg_nueva)
      arch_meg.write('\n')
```

heatmap.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

#cargar las librerías
import os
import sys
import re
import subprocess
import distutils.dir_util
import pandas as pd
from Bio import Phylo

#leer el árbol en formato Newick
tree = Phylo.read('/media/bioinfo/Sara/arbol_anib_nj', 'newick')
leaves = map(str, tree.get_terminals())
names=leaves

#leer el archivo en el que está la matriz y la base de datos del nombre del
archivo con el nombre de la cepa.
df = pd.read_csv('/home/bioinfo/Escritorio/perc_ids_7_modif.txt',sep='\t',
dtype=object, index_col=0)
bbdd_strain=open('/home/bioinfo/Escritorio/bbdd_nombre_cepa.txt','r')
dicc={}
dic_entero=[]
```

```
# por cada línea de la bbdd, separarla por los tabuladores y guardar el nombre
del archivo y la cepa.
for line in bbdd_strain:
      linea_n=line.rstrip('\n')
      linea=linea_n.split('\t')
      archivo,strain=linea[0],linea[1]

#por cada encabezamiento de columnas meter en un diccionario la cepa y su
identificador
      for i in range(len(df.columns)):
            cepa=str(df.columns[i])
            if archivo in cepa:
                  dicc[cepa]=strain
                  dic_entero.append(dicc)

#renombrar las columnas y los índices con el mismo nombre de cepa que tenían en
nuestra base de datos, ordenarlos en el mismo órden que el árbol de ANIb y
guardarlo en un csv.
df=df.rename(index=dicc, columns=dicc)
df.to_csv('/home/bioinfo/Escritorio/matriz_anib_cambionombre.csv', sep='\t',
index=True)
df=df.reindex(names)
df=df[names]
df.to_csv('/home/bioinfo/Escritorio/matriz_anib_cambioorden.csv', sep='\t',
index=True)
```

cambiar_nombre.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

#cargar las librerias
import os
import sys
import re
import numpy as np
import pandas as pd

#leer el archivo en el que están los Ortogrupos y abrir el archivo de bbdd de
los genomas de pseudomonas
df =
pd.read_csv('/home/bioinfo/Escritorio/Sara_Perez/temporthofinder/Orthogroups.csv
',sep='\t', dtype=object)
bbdd_strain=open('/home/bioinfo/Escritorio/Sara_Perez/genomas_pseudomonas_limpia
.csv','r')
dicc={}
dic_entero=[]

# por cada línea de la bbdd, separarla por las comas y sustituir la A por la F
para que las cepas se llamen GCF_XXXXXXXXX.X
for line in bbdd_strain:
      linea=line.split(',')
      archivo,strain=linea[7],linea[1]
      arch=re.sub('A','F',archivo)
      arc=re.sub(' ','',arch)

#por cada encabezamiento de columnas guardar en un diccionario la cepa y su
identificador
```

```
      for i in range(len(df.columns)):
            pepe=str(df.columns[i])
            if arc in pepe:
                  dicc[pepe]=arc
                  dic_entero.append(dicc)

#renombrar las columnas con el identificador GCF_XXXXXXXXXX.X y guardarlo en un
csv
df=df.rename(columns=dicc)
df.to_csv('/home/bioinfo/Escritorio/Sara_Perez/temporthofinder/Orthogroups_cambi
ado_nombre.csv', sep='\t', index=False)
```

csv2sql1.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

#cargar las librerias
import os
import sys
import re
import subprocess
import distutils.dir_util
import numpy as np
import pandas as pd

#abrir el archivo de los ortogrupos cambiados de nombre y leerlo con pandas
bbdd_orto='/home/bioinfo/Escritorio/Sara_Perez/temporthofinder/Orthogroups_cambi
ado_nombre.csv'
data=pd.read_csv(bbdd_orto,sep='\t', dtype=object)

#crear el archivo donde se creó la base de datos
arch_bbdd=open('/home/bioinfo/Escritorio/Sara_Perez/arch_bbdd.txt','w')

#recorrer la matriz y por cada término en la lista de datos, quedarse con el
identificador de la especie
for index,row in data.iterrows():
      for i in range(len(list(data))):
            iden=list(data)[i]
            if iden!='#':

#guardar el nombre del ortogrupo, el conjunto de proteínas de ese ortogrupo y el
identificador de la especie en un archivo nuevo
                  linea='\t'.join([str(row['#']),str(row[iden]),iden])
                  arch_bbdd.write(linea+'\n')
arch_bbdd.close()
```

csv2sql2.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

#cargar las librerias
import os
import sys
import re
import subprocess
```

```
import distutils.dir_util
import numpy as np
import pandas as pd

#abrir el fichero generado en el paso anterior y crear el nuevo fichero en el
que se guardará la nueva base de datos
arch_bbdd=open('/home/bioinfo/Escritorio/Sara_Perez/arch_bbdd.txt','r')
arch_bbdd_unica_linea=open('/home/bioinfo/Escritorio/Sara_Perez/arch_bbdd_una.tx
t','w')

# por cada línea del fichero guardar el identificador del ortogrupo, de la
especie y de la proteína
for line in arch_bbdd:
      linea=line.split('\t')
      ortogp,id_esp,id_prot=linea[0],linea[2],linea[1]

#separar los identificadores de proteína por la coma y por cada registro crear
una línea guardando en el archivo el ortogrupo, proteina y especie
      iden_prot=id_prot.split(',')
      for i in range(len(iden_prot)):
            print iden_prot[i]
            if iden_prot[i].startswith(' '):
                  proteina=re.sub('^ ','',iden_prot[i])
                  linea_bbdd='\t'.join([ortogp,proteina,id_esp])
                  arch_bbdd_unica_linea.write(linea_bbdd)
            else:
                  linea_bbdd_unica='\t'.join([ortogp,iden_prot[i],id_esp])
                  arch_bbdd_unica_linea.write(linea_bbdd_unica)
arch_bbdd_unica_linea.close()
arch_bbdd.close()
```

csv2sql3.py

```
#abrir el fichero anterior y crear el archivo en el que se guardará la base de
datos sin los valores 'nan'
arch_bbdd_unica_linea=open('/home/bioinfo/Escritorio/Sara_Perez/arch_bbdd_una.tx
t', 'r')
arch_bbdd_unica_limpia=open('/home/bioinfo/Escritorio/Sara_Perez/arch_bbdd_una_l
impia.txt','w')

#recorrer el fichero y si los identificadores de la proteína son distintos a
'nan' guardar esa línea en el nuevo archivo
for linea_bbdd in arch_bbdd_unica_linea:
      linea=linea_bbdd.split('\t')
      ortogp,id_prot,id_esp=linea[0],linea[1],linea[2]
      if id_prot!='nan':
            linea_def="\t".join([ortogp,id_prot,id_esp])
            arch_bbdd_unica_limpia.write(linea_def)
arch_bbdd_unica_limpia.close()
arch_bbdd_unica_linea.close()

#abrir el fichero anterior y crear el archivo en el que se guardará la base de
datos con el identificador corto de la proteína
arch_bbdd_unica_limpia=open('/home/bioinfo/Escritorio/Sara_Perez/arch_bbdd_una_l
impia.txt','r')
arch_bbdd_unica_limpia_corta=open('/home/bioinfo/Escritorio/Sara_Perez/arch_bbdd
_una_limpia_corta.txt','w')
```

```
#recorrer el fichero y si encuentra un identificador corto de proteína, guardar
este identificador en el nuevo archivo, junto con el ortogrupo y la especie.
for linea_bbdd_limpia in arch_bbdd_unica_limpia:
      linea=linea_bbdd_limpia.split('\t')
      ortogp,id_prot_largo,id_esp=linea[0],linea[1],linea[2]
      match=re.match('([A-Z]{2}_[0-9]+\.[0-9]{1}) (.*)',id_prot_largo)
      if match is not None:
            id_prot_corto=match.group(1)
            linea_bbdd_corta='\t'.join([ortogp,id_prot_corto,id_esp])
            arch_bbdd_unica_limpia_corta.write(linea_bbdd_corta)
arch_bbdd_unica_limpia.close()
arch_bbdd_unica_limpia_corta.close()
```

prot_comunes_grupo.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import os
import sys
import re
import subprocess
import distutils.dir_util
#mientras se esté en el rango de los seis grupos establecidos,crear la lista en
la que se guardarán todos los ortogrupos que están en todas las especies de ese
grupo y guardarlo en esa lista.
for i in range(6):
      lista_ortogrupo=[]

arch_orto_comun=open('/home/bioinfo/Escritorio/Sara_Perez/orto_comunes_grupo'+st
r(i+1)+'.csv','r')

arch_lista_orto_comun=open('/home/bioinfo/Escritorio/Sara_Perez/lista_orto_comun
es_grupo'+str(i+1)+'.txt','w')
      for linea_comun in arch_orto_comun:
            linea_n_comun=linea_comun.rstrip('\n')
            line_comun=linea_n_comun.split('\t')
            orto_comun=line_comun[0]
            lista_ortogrupo.append(orto_comun)
      arch_lista_orto_comun.write(str(lista_ortogrupo))
      arch_orto_comun.close()

#abrir el archivo en el que están todos los ortólogos que se encuentran en el
grupo y recorrer el fichero para guardar todos los ortogrupos, la especie y la
proteína.

arch_orto=open('/home/bioinfo/Escritorio/Sara_Perez/tabla_orto_grupo'+str(i+1)+'
.csv','r')
arch_prot_comunes=open('/home/bioinfo/Escritorio/Sara_Perez/prot_comunes_grupo'+
str(i+1)+'.csv','w')
      for line_orto in arch_orto:
            line_orto_n=line_orto.rstrip('\n')
            linea_orto=line_orto_n.split('\t')
            orto,esp, prot=linea_orto[0],linea_orto[1],linea_orto[3]

#recorrer la lista de ortogrupos comunes en el grupo y si cada elemento de la
lista coincide con el que está en el archivo donde están los ortogrupos, guardar
la proteína correspondiente en el nuevo archivo generado.
```

```
            for k in lista_ortogrupo:
                  if k!='ID_ORTO':
                        if k in orto:
                              arch_prot_comunes.write(prot+'\n')
      arch_orto.close()
      arch_prot_comunes.close()
```

genes_interesantes.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

#cargar las librerías
import os
import sys
import re
import subprocess
import distutils.dir_util

#abrir el archivo en el que están las especies y los archivos de salida de R,
recorrer el fichero y guardar el nombre de la especie y el grupo al que
pertenecen
arch_especies=open('/home/bioinfo/Escritorio/Sara_Perez/arch_esp_topGO.txt','r')

for line in arch_especies:
      linea_n=line.rstrip('\n')
      linea=linea_n.split('\t')
      especie,grupo=linea[0],linea[2]

#mientras se esté en el rango de los seis grupos definidos, crear la lista en la
que se guardarán todos los ortogrupos que están en todas las especies de ese
grupo y guardarlos en esa lista.
      for i in range(6):
            lista_ortogrupo=[]
            if str(i+1) in grupo:
                  arch_orto_comun= open
('/home/bioinfo/Escritorio/Sara_Perez/orto_comunes_grupo'+str(i+1)+'.csv','r')
                  for linea_comun in arch_orto_comun:
                        linea_n_comun=linea_comun.rstrip('\n')
                        line_comun=linea_n_comun.split('\t')
                        orto_comun=line_comun[0]
                        lista_ortogrupo.append(orto_comun)
                  arch_orto_comun.close()

#abrir el archivo en el que están todos los ortólogos que se encuentran en el
grupo y recorrer el fichero para guardar todos los ortogrupos, la especie y la
proteína.
                  arch_orto= open
('/home/bioinfo/Escritorio/Sara_Perez/tabla_orto_grupo'+str(i+1)+'.csv','r')
                  for line_orto in arch_orto:
                        line_orto_n=line_orto.rstrip('\n')
                        linea_orto=line_orto_n.split('\t')
                        orto,esp, prot=linea_orto[0], linea_orto[1],
linea_orto[3]
```

```
#recorrer la lista de ortogrupos comunes en el grupo y si cada elemento de la
lista coincide con el que está en el archivo donde están los ortogrupos y la
especie también coincide, entonces guardar la proteína correspondiente en el
nuevo archivo generado.
                        for k in lista_ortogrupo:
                              if k!='ID_ORTO':
                                    if k in orto and especie in esp:
                                          arch_int_genes = open
('/home/bioinfo/Escritorio/Sara_Perez/int_genes_grupo'+str(i+1)+'_'+esp+'.txt','
a')
                                          arch_int_genes.write(prot+'\n')
```

topGO.R

```
#Cargar la librería de topGO
library("topGO")

#Cargar las anotaciones de GO
annotation <-
readMappings(file="/home/bioinfo/Escritorio/Sara_Perez/PROTEINAS/EggNog/bbdd_ann
otatios_gi_go_limpia.txt")

#Preparar la lista de genes de la anotación
genenames <- names(annotation)

#Preparar la lista de los genes interesantes
interestingGenes <-
read.csv(file="/home/bioinfo/Escritorio/Sara_Perez/int_genes_grupo6_P.pseudoalca
ligenes_KF707.txt", header=TRUE)
myInterestingGenes<-c(as.vector(interestingGenes$proteinas))
geneList <-factor(as.integer(genenames %in% myInterestingGenes))
names(geneList) <- genenames

#Generador del objeto topGOdata (Proceso Biológico)
GOdataBP <- new("topGOdata", ontology = "BP", allGenes = geneList,
                annot = annFUN.gene2GO, gene2GO =annotation, nodeSize =2)
GOFisherBP <- runTest(GOdataBP, algorithm = "classic", statistic = "fisher")
go_test <- runTest(GOdataBP, algorithm = "weight01", statistic = "fisher")
GOKSBP <- runTest(GOdataBP, algorithm = "classic", statistic = "ks")
GOKSBP.elim <- runTest(GOdataBP, algorithm = "elim", statistic = "ks")
allresBP <- GenTable(GOdataBP, classicFisher = GOFisherBP,
                     classicKS = GOKSBP, elimKS =GOKSBP.elim,
                     orderBy = "classicFisher", ranksOf = "classicFisher",
topNodes = 100)
go_table <- GenTable(GOdataBP, weightFisher = go_test,
                     orderBy = "weightFisher", ranksOf = "weightFisher",
                     topNodes = sum(score(go_test) < .02))
showSigOfNodes(GOdataBP, score(GOFisherBP), firstSigNodes = 15, useInfo = "all")

#Guardar los resultados de la tabla en un archivo csv
write.csv(allresBP,
file="/home/bioinfo/Escritorio/Sara_Perez/GO_R/esp_6_3_topGO_BP.txt")

#topGOdata object generator (Mollecularfunction)

GOdataMF <- new("topGOdata", ontology = "MF", allGenes = geneList,
                annot = annFUN.gene2GO, gene2GO =annotation)
GOFisherMF <- runTest(GOdataMF, algorithm = "classic", statistic = "fisher")
```

```
GOKSMF <- runTest(GOdataMF, algorithm = "classic", statistic = "ks")
GOKSMF.elim <- runTest(GOdataMF, algorithm = "elim", statistic = "ks")
allresMF <- GenTable(GOdataMF, classicFisher = GOFisherMF,
                     classicKS = GOKSMF, elimKS =GOKSMF.elim,
                     orderBy = "classicFisher", ranksOf = "classicFisher",
topNodes = 100)

showSigOfNodes(GOdataMF, score(GOFisherMF), firstSigNodes = 15, useInfo = "all")
write.csv(allresMF,
file="/home/bioinfo/Escritorio/Sara_Perez/GO_R/esp_6_3_topGO_MF.txt")


#topGOdata object generator (Cellular)
GOdataCC <- new("topGOdata", ontology = "CC", allGenes = geneList,
                annot = annFUN.gene2GO, gene2GO =annotation)
GOFisherCC <- runTest(GOdataCC, algorithm = "classic", statistic = "fisher")
GOKSCC <- runTest(GOdataCC, algorithm = "classic", statistic = "ks")
GOKSCC.elim <- runTest(GOdataCC, algorithm = "elim", statistic = "ks")
allresCC <- GenTable(GOdataCC, classicFisher = GOFisherCC,
                     classicKS = GOKSCC, elimKS =GOKSCC.elim,
                     orderBy = "classicFisher", ranksOf = "classicFisher",
topNodes = 100)
showSigOfNodes(GOdataCC, score(GOFisherCC), firstSigNodes = 15, useInfo = "all")
write.csv(allresCC,
file="/home/bioinfo/Escritorio/Sara_Perez/GO_R/esp_6_3_topGO_CC.txt")
```

prot_diferenciales.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

#cargar las librerías
import os
import sys
import re
import subprocess

# abrir cada archivo donde están los ortólogos comunes en cada grupo y
guardarlos en una lista
arch_orto_comun_grupo1=open('/home/bioinfo/Escritorio/Sara_Perez/lista_orto_comu
nes_grupo1.txt','r')
for line1 in arch_orto_comun_grupo1:
      line_list1=eval(line1)
arch_orto_comun_grupo1.close()

arch_orto_comun_grupo2=open('/home/bioinfo/Escritorio/Sara_Perez/lista_orto_comu
nes_grupo2.txt','r')
for line2 in arch_orto_comun_grupo2:
      line_list2=eval(line2)
arch_orto_comun_grupo2.close()

arch_orto_comun_grupo3=open('/home/bioinfo/Escritorio/Sara_Perez/lista_orto_comu
nes_grupo3.txt','r')
for line3 in arch_orto_comun_grupo3:
      line_list3=eval(line3)
arch_orto_comun_grupo3.close()
```

```
arch_orto_comun_grupo4=open('/home/bioinfo/Escritorio/Sara_Perez/lista_orto_comu
nes_grupo4.txt','r')
for line4 in arch_orto_comun_grupo4:
      line_list4=eval(line4)
arch_orto_comun_grupo4.close()

arch_orto_comun_grupo5=open('/home/bioinfo/Escritorio/Sara_Perez/lista_orto_comu
nes_grupo5.txt','r')
for line5 in arch_orto_comun_grupo5:
      line_list5=eval(line5)
arch_orto_comun_grupo5.close()

arch_orto_comun_grupo6=open('/home/bioinfo/Escritorio/Sara_Perez/lista_orto_comu
nes_grupo6.txt','r')
for line6 in arch_orto_comun_grupo6:
      line_list6=eval(line6)
arch_orto_comun_grupo6.close()

#una vez que se obtiene cada lista, analizar cuales son los ortogrupos que están
en el grupo 1 y no en los demás, y así sucesivamente, y guardarlos en un
archivo.
arch_dif_orto=open('/home/bioinfo/Escritorio/Sara_Perez/lista_orto_diff_grupo.tx
t','w')
diferencia1=(set(line_list1))-(set(line_list2))-(set(line_list3))-
(set(line_list4))-(set(line_list5))-(set(line_list6))
arch_dif_orto.write(str(diferencia1)+'\n')

diferencia2=(set(line_list2))-(set(line_list3))-(set(line_list4))-
(set(line_list5))-(set(line_list6))-(set(line_list1))
arch_dif_orto.write(str(diferencia2)+'\n')

diferencia3=(set(line_list3))-(set(line_list4))-(set(line_list5))-
(set(line_list6))-(set(line_list1))-(set(line_list2))
arch_dif_orto.write(str(diferencia3)+'\n')

diferencia4=(set(line_list4))-(set(line_list5))-(set(line_list6))-
(set(line_list1))-(set(line_list2))-(set(line_list3))
arch_dif_orto.write(str(diferencia4)+'\n')

diferencia5=(set(line_list5))-(set(line_list6))-(set(line_list1))-
(set(line_list2))-(set(line_list3))-(set(line_list4))
arch_dif_orto.write(str(diferencia5)+'\n')

diferencia6=(set(line_list6))-(set(line_list1))-(set(line_list2))-
(set(line_list3))-(set(line_list4))-(set(line_list5))
arch_dif_orto.write(str(diferencia6)+'\n')

#una vez obtenidos los ortólogos de cada grupo, se extrajeron las secuencias de
las proteínas ortólogas, para ello se abrió el archivo donde estaban recogidos
estos ortólogos diferentes
arch_orto_espec=open('/home/bioinfo/Escritorio/Sara_Perez/lista_orto_diff_grupo.
txt','r')

#mientras se está en el rango de los 6 grupos, crear el archivo de prteínas
específicas de cada grupo y acceder a la base de datos de todos los ortogrupos
pertenecientes a ese grupo
i=0
for line in arch_orto_espec:
```

```
      i+=1
      line_list=eval(line)
      lista_prot=[]

arch_especif_prot_grupo=open('/home/bioinfo/Escritorio/Sara_Perez/lista_prot_esp
ecifico_grupo'+(str(i))+'.txt','w')

arch_orto=open('/home/bioinfo/Escritorio/Sara_Perez/tabla_orto_grupo'+str(i)
+'.csv','r')

#por cada línea, guardar el ortogrupo y la proteína, y si los ortogrupos
coinciden y la proteína no está en la lista de proteínas, guardarlo en el
archivo para las proteínas específicas.
      for line_orto in arch_orto:
            line_orto_n=line_orto.rstrip('\n')
            linea_orto=line_orto_n.split('\t')
            orto,prot=linea_orto[0],linea_orto[3]
            for k in line_list:
                  if k!='ID_ORTO':
                        if k in orto:
                              if prot not in lista_prot:
                                    lista_prot.append(prot)
                                    arch_especif_prot_grupo.write(prot+'\n')
arch_especif_prot_grupo.close()
arch_orto.close()
arch_orto_espec.close()
```

muestreo_aleatorio.py

```
# se crea el directorio en el que se van a guardar todos los análisis de topGO y
el archivo donde se van a guardar las 200 replicaciones del muestreo aleatorio
de genomas.
directorio_orto_species='/home/bioinfo/Escritorio/Sara_Perez/GO_R/'
file_list_orto_sp=os.listdir(directorio_orto_species)
arch_lista_desor=open('/home/bioinfo/Escritorio/Sara_Perez/lista_desor_arch_orto
_esp.txt','w')

#realizar el muestreo aleatorio de las bacterias con 200 replicaciones y lo
guardarlo en un archivo
for i in range(200):
      random.shuffle(file_list_orto_sp)
      #print 'longitud',len(file_list_orto_sp)
      arch_lista_desor.write(str(file_list_orto_sp)+'\n')
arch_lista_desor.close()
arch_lista_desor=open('/home/bioinfo/Escritorio/Sara_Perez/lista_desor_arch_orto
_esp.txt','r')
```

collector_curve.R

```
#R Script para generar muestras a partir de una tabla de ortólogos y calcular la
Curva del Coleccionista.
#En nuestro caso, leeremos directamente la tabla.
ortologos <-
read.table("/home/bioinfo/Escritorio/Sara_Perez/arch_orto_especie.txt",
header=T, sep="\t")

# Cálculo de la média y el intervalo de confianza (CI) al 95%
```

```
#Primero tenemos que asignar cada fila a un vector para posteriormente calcular
la media y la desviación estándar.
#Posteriormente, usaremos el calculo de la desviación estándar, junto con el
tamaño muestral (en nuestro caso 200) para obtener el CI y todo esto lo
guardamos en vectores para posteriormente representarlo.
#Solo comento lo que he hecho para el primer caso:

output = NULL # Creamos una variable vacía (NULL). Es obligatorio que esté antes
del bucle.
for (i in 1:359) { # Bucle for desde 1 a 356 (número de filas en).
  row <- t(otu[i,]) #Creamos el vector ""row"" que resulta de los valores
transpuestos (usando la función t()) de los valores de otu95samp[i,], --> toma
cada fila (desde 1 a 356) y la convertimos en una columna.
  mean <- mean(row) #Calculamos la media de el vector row.
  error <- qt(0.975,df=200-1)*sd(row)/sqrt(200) #Esta fórmula calcula el error
del vector row. Donde: 0.975 = intervalo de confianza del 95%, df = grados de
libertad, sd = desviación típica y 200 = tamaño muestral. Fórmula valida si
sigue una distribución t-student. Si la distribución fuese normal:
qnorm(0.975)*sd(row)/sqrt(200)
  lci <- mean-error # Parte inferior del intervalo de confianza.
  hci <- mean+error # Parte superior del intervalo de confianza.
  ncepa = i #variable ncepa como número de la fila que estamos estudiando.
Unicamente nos sirve para luego añadirla a la matriz final.
  output <- rbind(output, data.frame(ncepa, mean,lci,hci)) #Escribe en la
variable NULL creada anteriormente la combinación de filas (función rbind()) de
la variable NULL junto con la media, lci y hci.
}

#Como resultado, output es una tabla que contiene las 356 filas y 4 columnas:
número de la cepa (típicamente el X para la representación) y las 3 columnas:
mean, lci y hci (que serían la Y para representarlas).

# ---- DIAGRAMA
#Para representar todo esto en un diagrama:
plot(output$ncepa, output$mean, type="l", xlab = "Nº genomas de la muestra",
ylab = "Nº ortogrupos observados", col = "2", xlim=c(0,368), ylim=c(0,16000))
     lines(output$ncepa, output$lci, col="#DC6D6D", lty=5)
     lines(output$ncepa, output$hci, col="#E6A4A4", lty=6)
```